Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería Informática

Proyecto de Fin de Carrera

Resolución del Rush Hour y representación tridimensional

miércoles, 03 de octubre de 2012

Autor: Alberto López Cebrián
Tutor: Carlos Linares López

“La inteligencia artificial nunca podrá competir con la estupidez natural.”

Anónimo

Agradecimientos

Quiero agradecer el apoyo recibido por todas las personas cercanas a mí y que me han animado a hacer un buen proyecto de fin de carrera.

A mi madre, mi padre y mi hermano por haberse preocupado desde el principio por mí y por mi proyecto y, de este modo, apoyándome para realizar un proyecto digno de ingeniería.

A mis compañeros y amigos de la universidad y de fuera de ella, que me han proporcionado las risas necesarias para coger la carrera y el proyecto con más ganas durante los días de trabajo.

A todos los profesores de la carrera, que me han aportado todos los conocimientos que tengo en la actualidad.

Y, finalmente, a mi tutor Carlos Linares López, que sin su ayuda y apoyo no habría conseguido encontrar la motivación suficiente para realizar este proyecto.

Alberto López Cebrián

Madrid, 12 de julio de 2012

Índice

Índice de ilustraciones

Índice de tablas

Capítulo 1

Introducción

El considerable avance tecnológico de las últimas décadas ha ido generando nuevas inquietudes en los campos de investigación y, entre ellos, una gran parte de la atención se centra en la Inteligencia Artificial, buscando la posibilidad de crear sistemas capaces de resolver problemas de carácter general.

En la actualidad se ha conseguido desarrollar sistemas capaces de resolver determinados problemas dentro de un mismo dominio. Estos sistemas están basados en técnicas de búsqueda que consisten en algoritmos no informados o, en el caso de querer mejorar el tiempo en el que se encuentra la solución y el consumo de recursos, algoritmos informados con heurística. También es cierto que se ha conseguido realizar técnicas no dependientes del dominio aunque realmente no se puede utilizar eficientemente en todas las situaciones.

El ámbito donde se aplican estas técnicas es muy amplio, desde juegos de todo tipo: puzzle (15 Puzzle[1], Rush Hour[2]), estrategia (Reversi[3], Ajedrez[4]), simulación (Forza Motorsport[5], Ace Combat[6]), etc; hasta proyectos a gran escala como el Mars Pathfinder[7] enviado a Marte a finales de 1996 por la NASA. Sin embargo, este proyecto se ha centrado en la realización de un sistema que resuelva cualquier instancia del juego Rush Hour, de forma óptima, e intentar reducir al máximo la cantidad de recursos utilizados.

Rush Hour (Hora Punta como traducción literal) es un juego de puzzle que trata de un tablero de 6x6 casillas en el que se dispone de vehículos de 1x2 ó de 1x3 casillas, y en el que hay que conseguir extraer un vehículo de color rojo que ha quedado atrapado en el atasco de coches, mediante el deslizamiento de los vehículos.

Ilustración 1 – Rush Hour

Además, el juego tiene unas reglas básicas que restringen el movimiento de los vehículos. Éstos sólo pueden moverse en dos direcciones (hacia adelante y hacia atrás) sin salirse del tablero y en ningún caso girar. Tampoco podrá haber en ningún momento más de un vehículo en una misma casilla, lo que implica que éstos no podrán moverse a través de los demás vehículos.

La forma de representar gráficamente los resultados obtenidos en la resolución de cada uno de los tableros, se ha realizado mediante una interfaz tridimensional, incluyendo un editor de tableros, los cuales se podrá guardar para resolverlos posteriormente con el algoritmo de búsqueda utilizado.

Todo esto se explica con detalle en los siguientes capítulos de esta memoria, los cuales están organizados de la siguiente forma:

Capítulo 2 – Estado de la cuestión: En este apartado se mostrará cómo funciona la inteligencia artificial aplicada a los juegos de puzzle, concretamente al Rush Hour, y cuáles son los diferentes algoritmos de búsqueda que se pueden utilizar, sus ventajas y sus inconvenientes.

Capítulo 3 – Objetivos: En este capítulo se listará una serie de objetivos que se han cumplido en la realización de este proyecto así como su explicación de forma breve y concisa.

Capítulo 4 – Desarrollo: En este cuarto apartado, se explicará detalladamente cada uno de los pasos realizados para el desarrollo del proyecto, tanto para la interfaz gráfica como para la parte de inteligencia artificial. Además, se expondrán, en una sección dedicada a ello, los distintos problemas encontrados a la hora de implementar el proyecto y la forma en la que se han resuelto.

Capítulo 5 – Pruebas y resultados: Una vez realizada la implementación del proyecto, se comprobará el algoritmo de búsqueda con distintas heurísticas. Los resultados de estas pruebas se incluirán en esta sección junto con unas conclusiones con referencia a dichos resultados.

Capítulo 6 – Conclusiones: En esta sección se comprueba si el proyecto ha alcanzado los objetivos descritos en el capítulo 3 de esta memoria.

Capítulo 7 – Líneas futuras: Normalmente un proyecto suele incluir nuevas funcionalidades en próximas actualizaciones y, en este apartado, se expondrá algunas posibles líneas futuras a las que orientar este proyecto.

Capítulo 2

Estado de la cuestión

Desde tiempos inmemoriables, los juegos han causado una especial atracción hacia las personas ya sea por el entretenimiento que les proporciona o por el hecho de mejorar o poner a prueba el razonamiento lógico de uno mismo. Estos juegos podían ser tanto unipersonales, es decir, juegos de un jugador en el que el objetivo normalmente es conseguir llegar a una meta desde un tablero inicial; o juegos con adversario en el que el objetivo consiste en ganar al contrincante.

La idea de que las computadoras puedan jugar a estos juegos está presente desde la existencia de las mismas, comenzando en el siglo XIX cuando Babbage, arquitecto de computadores, pensó en la posibilidad de programar una máquina analítica que fuera capaz de jugar al ajedrez y, actualmente esa idea ha evolucionado drásticamente hasta llegar a los juegos actuales y su inteligencia artificial.

Rush Hour

Rush Hour es un juego creado a finales de los años 70 por Nob Yoshigahara[8]. El juego simula un atasco de vehículos en un tablero de 6x6 casillas del que uno de los coches tendrá que salir.

Actualmente el juego está fabricado por la empresa ThinkFun[9], fundada en 1985 por Bill Ritchie y Andrea Barthello, y que comenzó fabricando 4 juegos, inventados por un amigo de la familia, llamados: Spin-out, The Cat, The Horse y Hexadecimal Puzzle.

Ilustración 2 - Logo de la empresa ThinkFun

Más tarde, en 1996, ThinkFun sacó al mercado el juego más exitoso de esta empresa, el Rush Hour. En la primera versión, se introdujo un pack de 40 tableros ordenados. Sin embargo, la empresa ha fabricado diversas expansiones, con nuevas barajas (incluyendo mayores niveles de dificultad) e incluso un cambio de diseño en los vehículos. Además, recientemente la empresa ha creado una aplicación para iOS y Android en la que se puede jugar al juego.

Ilustración 3 - Aplicación de Rush Hour para Android

Rush Hour se puede generalizar ampliando el tablero de forma ilimitada (NxN) e incluso con dimensiones distintas para los lados (NxM). El decidir si el tablero tiene solución o no se trata de un problema NP-Completo, lo que en términos de complejidad computacional significa que son los problemas más difíciles del espacio de problemas NP. Estos problemas NP-Completo pertenecen al subconjunto de NP que tienen menos posibilidades de estar en P, es decir, de tener un tiempo polinómico para su resolución.

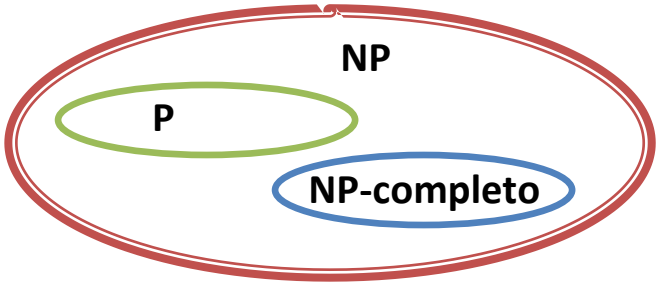


Ilustración 4 - Espacio de problemas NP

Reglas Rush Hour

Al tratarse de un Sliding Block Puzzle, el jugador es retado a llegar a un estado final simplemente deslizando los vehículos y sin levantarlos del tablero en ningún caso, siguiendo además una serie de restricciones.

Además, si el tablero tiene solución, se puede llegar desde cualquiera de los estados finales posibles hasta el estado inicial con la secuencia invertida de movimientos.

Ilustración 5 - Transición de un estado inicial a un estado final

Vehiculos

Las piezas de este juego consisten en tres tipos de vehículos, aunque en algunas versiones (p.e. para móviles) pueden aparecer simplemente tres tipos de rectángulos. En este proyecto hablaremos de vehículo y éstos son los tres tipos existentes:

Vehículo principal: Se trata del vehículo que hay que sacar del tablero para considerar que se ha obtenido un nodo meta. Este es un vehículo de color rojo y de tamaño 2x1 con orientación horizontal y con la variable de posicionamiento y fijada en y = 3.

Vehiculos: Se trata de vehículos de diferentes colores y de tamaño 2x1 con cualquier orientación y posición dentro de los siguientes límites:

- Si el vehículo está orientado horizontalmente, la posición puede ser cualquiera contenida en x = [0, 4], y = [0, 5].
- En caso contrario, si la orientación es vertical, la posición del vehículo varía entre x = [0, 5], y = [0, 4].

Camiones: Se trata de vehículos de diferentes colores y de tamaño 3x1 con cualquier orientación y posición dentro de los siguientes límites:

- Si el vehículo está orientado horizontalmente, la posición puede ser cualquiera contenida en x = [0, 3], y = [0, 5].

- En caso contrario, si la orientación es vertical, la posición del vehículo varía entre x = [0, 5], y = [0, 3].

Una consideración importante es la imposibilidad de alcanzar un nodo meta en caso de que haya un vehículo orientado de forma horizontal en $x > x_{principal}$, y = 3.

Restricciones

En este juego únicamente existe un tipo de restricción y es la restricción de movimiento. Esta restricción consiste en los siguientes puntos:

- El movimiento de los vehículos sólo puede ser hacia adelante o hacia atrás y de una única casilla cada vez. En ningún caso se podrá girar o mover hacia los laterales ningún vehículo por lo que restringe a 2 el número máximo de posibles movimientos para cada vehículo.
- En ningún momento podrá permanecer más de un vehículo en una misma casilla del tablero y, por ende, no se puede traspasar vehículos con otros vehículos.
- Para considerar un nodo meta, el vehículo principal ha de encontrarse en la posición x = 4, y = 3 del tablero.

NOTA: en algunas versiones del Rush Hour se considera nodo meta cuando el vehículo principal se encuentra totalmente fuera del tablero, es decir, en x = 6, y = 3. Sin embargo, dado que una vez que el vehículo principal se encuentre en la posición x = 4, y = 3 es trivial llegar a la solución de las otras versiones del juego, no se ha considerado contar el resto de movimientos hasta llegar a x = 6.

Tableros

En la versión original del juego de mesa, se incluye un conjunto de 40 cartas, cada una de las cuales indica un tablero distinto y una de sus soluciones, que utiliza el menor número de movimientos posibles, por detrás. Dichas cartas están divididas en 4 bloques de 10, diferenciados por su nivel de dificultad.

El nivel de dificultad depende del número mínimo de movimientos necesarios para resolver el tablero. Los 4 niveles son los siguientes:

- Principiante: entre X y Y movimientos.
- Intermedio: entre X y Y movimientos.
- Difícil: entre X y Y movimientos.
- Experto: entre X y Y movimientos.

Inteligencia Artificial aplicada al Rush Hour

Para todos los juegos unipersonales de tipo puzle existe la forma de encontrar su solución (si la tiene) mediante una computadora. Pero para ello es muy importante tener un concepto muy claro de lo que es el juego en sí, de sus elementos, sus restricciones y, lo más importante, una forma de representar cada uno de los estados, ya que esto último nos servirá para que la computadora sepa en qué situación se encuentra el problema.

Básicamente, cada instancia de un juego tiene un estado inicial y uno o varios estados finales. Por ejemplo:

- En el 15-puzzle tenemos un único estado final que es el que se muestra a continuación:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Ilustración 6 - Estado final del 15-puzzle

- Sin embargo, en el juego del Rush Hour nos podemos encontrar con muchos estados finales, aunque si lo que queremos es la solución óptima (el mínimo número de movimientos), siempre tendremos el mismo estado final para cada uno de los tableros.

Dificultades

Para una persona, averiguar que hay que empeorar la situación en la que se encuentra el problema para llegar hasta su solución es uno de las dificultades mayores a la hora de enfrentarse a este tipo de retos. El Rush Hour es un juego en el que esta necesidad es muy frecuente, sobre todo en tableros con un alto nivel de dificultad, en los que habrá que avanzar y retroceder varias veces.

Sin embargo, para un computador, retroceder es un movimiento más que comprobar por lo que, en principio, no tiene mayor dificultad. Una de las mayores dificultades de este juego es la necesidad de muchas veces tener que empeorar la situación para conseguir resolver el tablero, sobre todo en los niveles de mayor dificultad.

Para una persona puede resultar complicado aunque en muchas ocasiones se puede comprobar la necesidad de "ir hacia atrás" sin mucha dificultad. Sin embargo, para un computador, esto implica un tiempo de búsqueda mucho mayor, ya que la heurística utilizada devolvería un peor valor cuando se empeora la situación. De esto se hablará en el capítulo 4, en la sección de heurísticas.

Algoritmos de búsqueda

Actualmente, existen diversos algoritmos de búsqueda[10] para la resolución de instancias de juegos de puzle unipersonales, así como otros algoritmos orientados a un agente que juegue contra una persona (u otro agente), etc.

Estos algoritmos de búsqueda están basados en un árbol de nodos que se van recorriendo y expandiendo hasta alcanzar un estado final o meta. Cada nodo puede tener varios descendientes y existe, además, la posibilidad de llegar a un nodo desde varios estados anteriores por lo que, normalmente, la búsqueda también se puede representar mediante un grafo.

Además, normalmente los movimientos son invertibles, es decir, si desde el nodo A podemos ir hasta el nodo B mediante un movimiento M, se puede llegar al nodo A desde el nodo B mediante el movimiento inverso M-1.

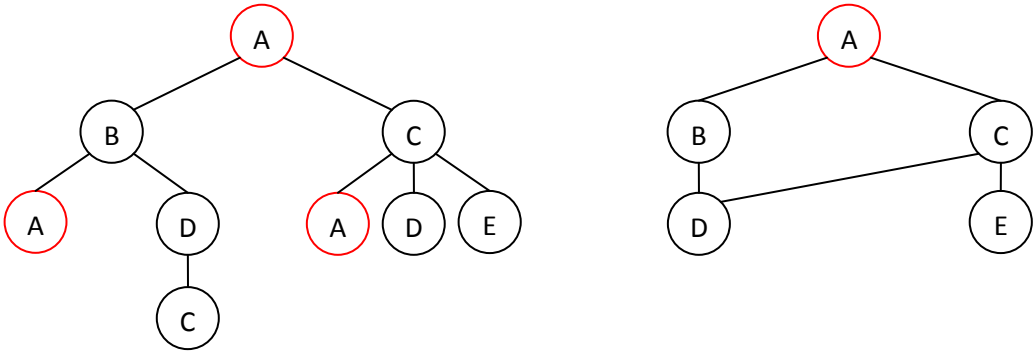


Ilustración 7 - Izq) Un árbol de nodos. Der) El mismo árbol representado por un grafo

Si consideramos un árbol como representación de los nodos, el número de movimientos que hay que realizar para llegar desde el estado inicial hasta el estado final es el número de nivel donde se ha encontrado el nodo meta. En esta imagen podemos comprobarlo con un ejemplo:

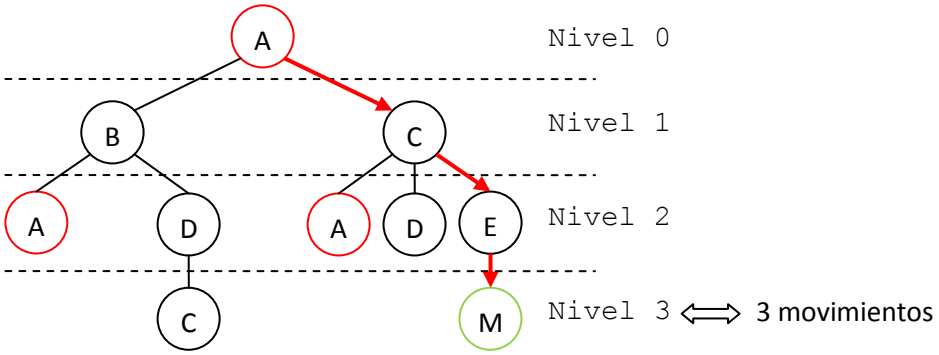


Ilustración 8 - El número de movimientos depende de la profundidad de la meta

Normalmente, para comparar resultados entre algoritmos de búsqueda o entre varias heurísticas utilizadas en un mismo algoritmo de búsqueda se utilizan cuatro medidas:

- **Tiempo**: Es el tiempo de cómputo o tiempo que se tarda en encontrar una solución (o comprobar que no tiene solución si se da el caso).

- **Número de nodos expandidos**: Es la cantidad de nodos de los que se obtienen sus hijos para comprobarlos.

- **Número de nodos generados**: Es el número total de nodos que se han generado, resultantes de la expansión de su nodo sucesor.

- **Número de pasos hasta la meta**: Es la altura del árbol a la que se ha encontrado un nodo solución. En algunos casos este resultado no es óptimo (mínimo número de pasos) por lo que es bueno tener esta medida en cuenta.

En las pruebas realizadas con el algoritmo y heurísticas implementadas en este proyecto se ha realizado una comprobación de estos resultados, comparando varias heurísticas sencillas sobre el algoritmo IDA* explicadas posteriormente. En el capítulo 5 de pruebas se puede comprobar la batería de pruebas realizadas y sus resultados.

Los algoritmos de búsqueda se pueden dividir en dos grandes grupos:

Algoritmos de búsqueda no informada

Estos algoritmos están orientados a una búsqueda exhaustiva y sin información sobre el medio en el que nos encontramos por lo que, normalmente serán más ineficientes que el otro tipo de algoritmos de búsqueda que veremos más adelante ya que se tendrá que comprobar una cantidad de nodos superior sin dar prioridad a ninguno.

Debido a la ineficiencia de estos algoritmos, simplemente se ha procedido a enumerar y aportar una breve explicación de los dos algoritmos "típicos" de esta rama:

- **Primero en amplitud**: Es un algoritmo que siempre encontrará (si existe) la solución óptima ya que observa todos los hijos de un nodo antes de bajar al siguiente nivel.

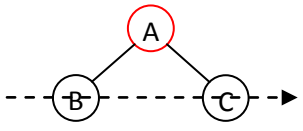


Ilustración 9 - Resumen del algoritmo de primero en amplitud

- **Primero en profundidad**: Este algoritmo es más rápido aunque no asegura la solución óptima. De hecho, tampoco asegura encontrar una solución ya que hay que aplicar un umbral máximo de profundidad.

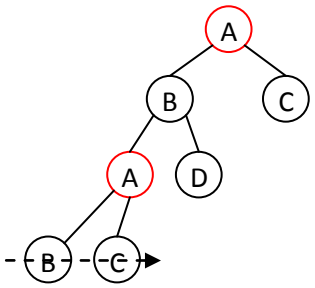


Ilustración 10 - Resumen del algoritmo de primero en profundidad

Estos algoritmos están orientados a una búsqueda más precisa mejorando, normalmente, tanto el tiempo de búsqueda como el uso de recursos de memoria del computador. Para ello, es necesario conocer el medio y obtener información que ayude a decidir el nodo que tiene mayor preferencia de ser expandido. Este conocimiento del medio se obtiene mediante unas funciones llamadas heurísticas y que se explican en detalle más adelante.

A continuación se expone una explicación de algunos de los algoritmos más importantes y relevantes para este proyecto.

Primero el mejor

El funcionamiento de este algoritmo consiste en una búsqueda guiada mediante una heurística de tal forma que se expande primero el mejor de los nodos.

1- Se expande el mejor de los nodos aún no expandidos obteniendo sus sucesores. El mejor nodo se obtendrá de la lista "abiertos" que tenga un menor f(nodo), siendo:

$$f(nodo) = h(nodo)$$

2- Se obtienen sus sucesores y se comprueba si alguno de ellos es un nodo meta:
   a. Si es meta se finaliza con éxito.
   b. Si no, se comprueba si alguno de los hijos ya se habían generado o expandido. En cualquiera de los dos casos se elimina el hijo. Después se ordenan dependiendo de su f(nodo) en la lista de nodos generados.
3- Repetir el paso 1 hasta encontrar una meta o no disponer de más nodos generados (cuyo caso supondría un resultado fallido).

Veamos un ejemplo para comprender el funcionamiento de este algoritmo:

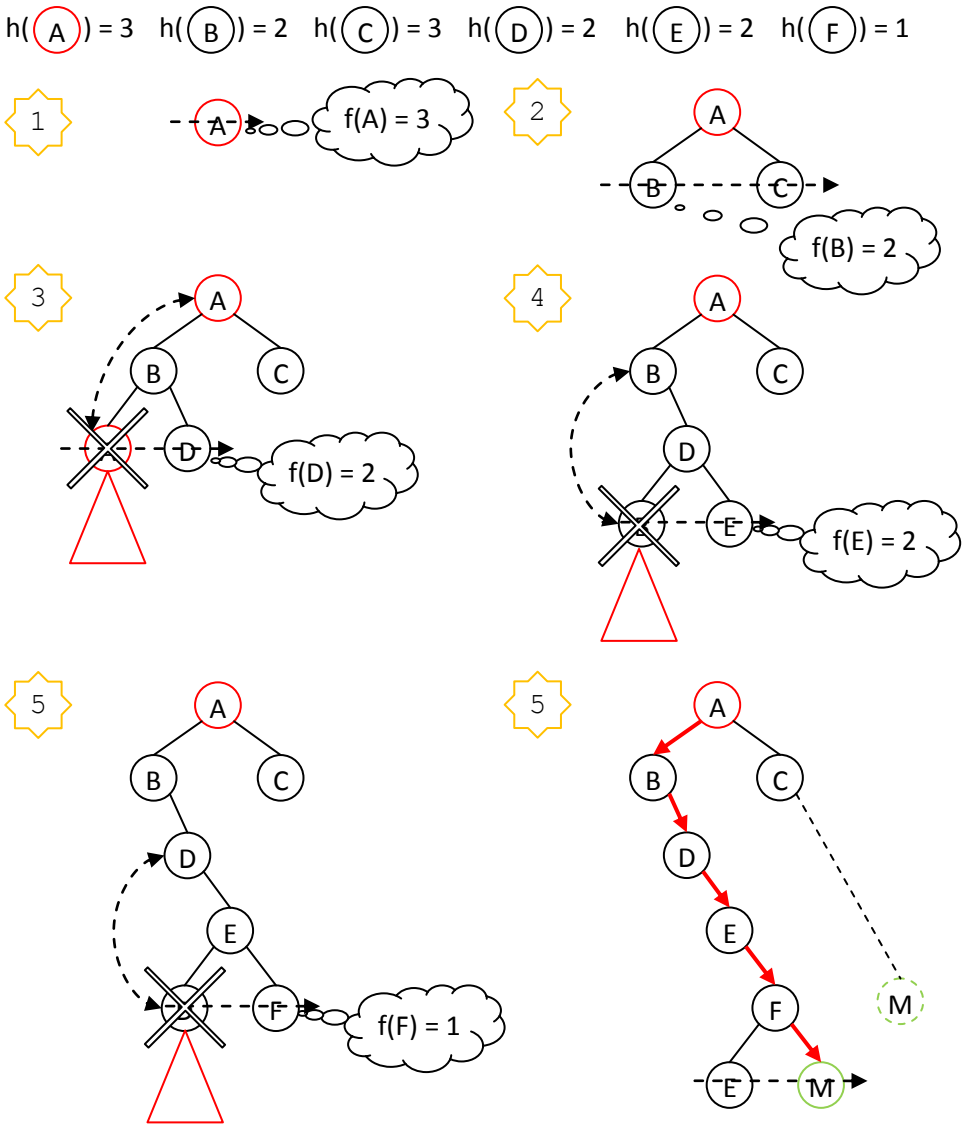


Ilustración 11 - Ejemplo del algoritmo Primero el mejor

Observando el ejemplo anterior podemos describir las siguientes conclusiones.

Características

- Es un algoritmo completo ya que encontrará solución siempre que exista.
- No es óptimo ya que podría encontrar primero una solución que no fuese óptima.
- Es un algoritmo dependiente de la heurística, encontrará la solución antes o después dependiendo de la calidad de ésta.

Ventajas

- Normalmente este algoritmo es muy rápido en ejecutarse aunque el tiempo también depende de la eficiencia y eficacia de la heurística.

Inconvenientes

- Podría encontrar una solución no óptima ya que no se tiene en cuenta el coste de llegar hasta cierto nodo sino tan solo el coste heurístico (en el ejemplo podemos observar que se podría haber llegado al nodo meta con un movimiento menos que los que se han hecho).
- Si la heurística es mala se aproximará al tiempo que tarda el algoritmo de primero en amplitud.
- El consumo de memoria es alto y, si la heurística es mala, el consumo se acerca al mismo que primero en amplitud.

*Algoritmo A**

El funcionamiento de este algoritmo es muy similar al anterior ya que se expande primero el mejor de los nodos pero, además, se añade una penalización por coste del camino para añadir un "componente de anchura" y así explorar por otros caminos.

1- Se expande el mejor de los nodos aún no expandidos obteniendo sus sucesores. El mejor nodo se obtendrá de la lista "abiertos" que tenga un menor f(nodo), siendo:

$$f(nodo) = g(nodo) + h(nodo)$$

El valor h(nodo) nos lo dará la heurística que se utilice para el algoritmo.

2- Comprobamos cada sucesor para comprobar uno de los siguientes tres casos:
   a. Si ya existía un nodo previamente expandido que sea igual al sucesor: nos quedamos con el camino de menor coste g(nodo).
   b. Si ya existía un nodo previamente generado (no expandido) que sea igual al sucesor: nos quedamos con el camino de menor coste g(nodo).
   c. Si el sucesor no había sido ni expandido ni generado previamente: se añade a la lista de nodos generados (abiertos).
3- Este proceso se repite hasta alcanzar el nodo meta.

Veamos un ejemplo para comprender el funcionamiento de este algoritmo:

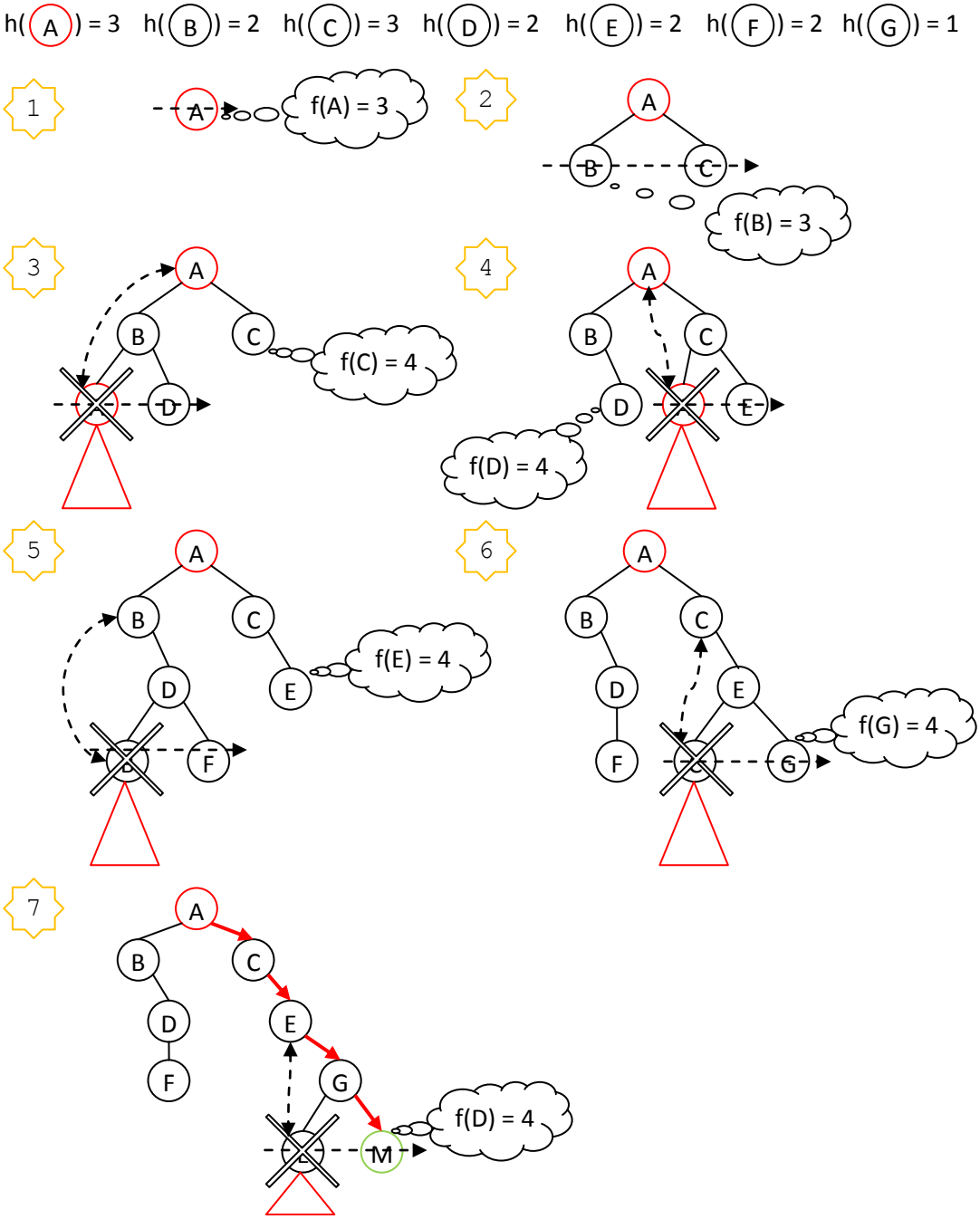


Ilustración 12 - Ejemplo del algoritmo A*

Observando el ejemplo anterior podemos describir las siguientes conclusiones.

Características

- Es un algoritmo completo.
- Es un algoritmo óptimo, es decir, siempre encontrará la solución óptima.
- Es un algoritmo dependiente de la heurística, encontrará la solución antes o después dependiendo de la calidad de ésta.

Ventajas

- Normalmente este tipo de algoritmos es mucho más rápido que los algoritmos no informados y, en concreto, el A* es un algoritmo muy rápido aunque también depende de la heurística que utilice.

Inconvenientes

- Si la heurística es mala se aproximará al tiempo que tarda el algoritmo de primero en amplitud.
- El consumo de memoria es alto y, si la heurística es mala, el consumo se acerca al mismo que primero en amplitud.

*Algoritmo IDA**

El funcionamiento de este algoritmo consiste en una búsqueda que se reinicia cuando no hay más nodos que expandir con coste igual o inferior a un umbral. Cada vez que se reinicia, se aumenta dicho umbral. Esto evita el consumo exponencial de memoria, convirtiéndolo en un consumo lineal (o constante para una implementación muy eficiente). El procedimiento es el siguiente:

1- El umbral inicial se establece como el coste heurístico del nodo inicial:

$$\mu = h(nodo)$$

El valor h(nodo) nos lo dará la heurística que se utilice para el algoritmo.

2- Se expande el nodo inicial.

3- Se comprueba mediante primero en profundidad los nodos generados y se expanden aquellos en los que se cumple la siguiente restricción:

$$h(nodo) + g(nodo) \leq \mu$$

4- Una vez se agoten los nodos generados que cumplen dicha restricción, se establece:

$$\mu = \min\big(h(nodo) + g(nodo)\big)$$

5- Se vuelve al paso 2 y se reitera hasta encontrar la solución.

Este algoritmo puede ser implementado de varias maneras. Una de ellas es generar todos los hijos de un nodo a la vez, comprobar si alguno es nodo meta y después expandirlos de uno en uno (siempre que no exceda su coste del umbral como se ha explicado antes) repitiendo el proceso y borrando los nodos de la memoria cada vez que se vaya hacia atrás. Esta manera es muy ineficiente en cuanto a recursos de memoria.

También se puede implementar de forma que sólo un nodo sea guardado en memoria (además de información adicional que se explica en apartados posteriores). Esto se consigue mediante la aplicación de una función para expandir y la función inversa para "contraer" el nodo.

Veamos un ejemplo de ambos para comprender el funcionamiento de este algoritmo:

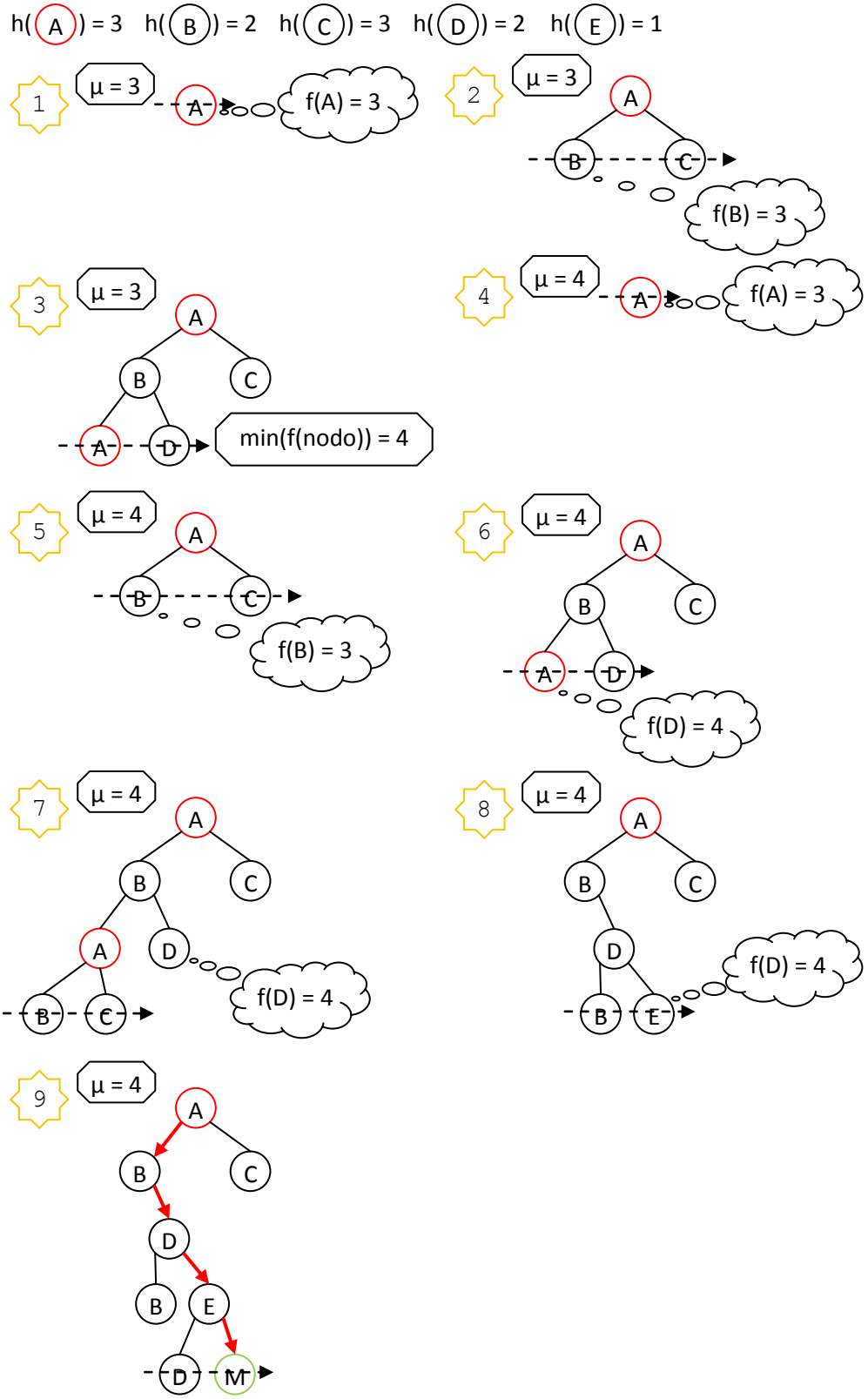


Ilustración 13 - Ejemplo 1 del algoritmo IDA*

En este ejemplo se puede observar que se encuentra la solución de manera rápida aunque el consumo de recursos es elevado. A continuación se muestra un ejemplo de la forma más correcta de implementar este algoritmo (se supone la misma disposición de nodos).

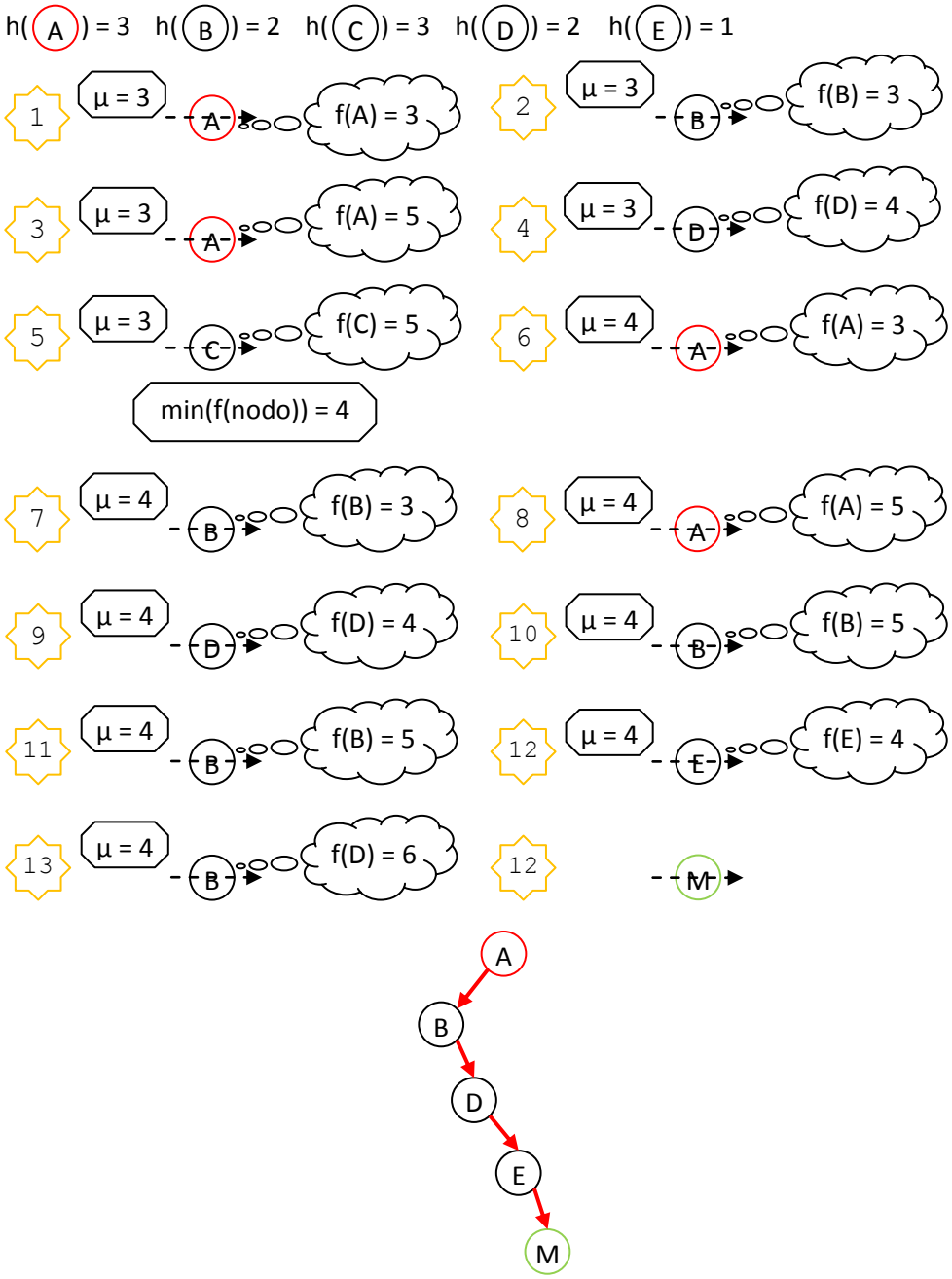


Ilustración 14 - Ejemplo 2 del algoritmo IDA*

Como se puede observar, esta implementación es mucho más eficiente en cuanto a consumo de memoria se refiere, llegando a la misma solución que la otra implementación.

Con estos ejemplos podemos describir las siguientes conclusiones.

Características

- Es un algoritmo completo.
- Es un algoritmo óptimo, es decir, siempre encontrará la solución óptima.
- Es un algoritmo dependiente de la heurística, encontrará la solución antes o después dependiendo de la calidad de ésta.

Ventajas

- Es un algoritmo rápido, aunque depende también de la heurística.
- Es un algoritmo que resolverá el problema sea lo grande que sea, ya que el consumo de memoria sería constante.
- Encontrará, siempre que exista, la solución óptima.

Inconvenientes

- Es más lento que otros algoritmos informados como el A*.

Mejoras aplicables a IDA*

Uno de los mayores problemas del algoritmo IDA* es que no guarda información sobre los nodos ya visitados, por lo que convendría mejorar éste algoritmo evitando la navegación repetida a dichos nodos. Estas mejoras pueden ser:

- Evitar regresar al padre.

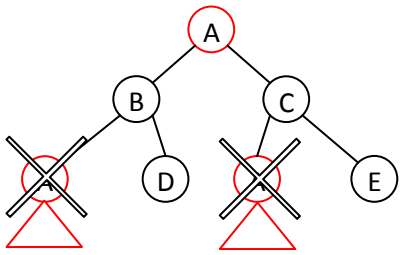


Ilustración 15 - Ejemplo de evitar volver al nodo padre

- Evitar expandir cualquier nodo previamente expandido (incluso en otras ramas del árbol).

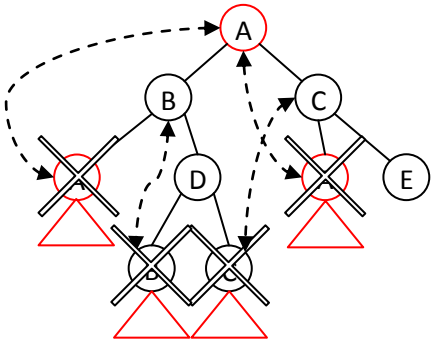


Ilustración 16 - Ejemplo de evitar expandir cualquier nodo previamente expandido

- Ambas opciones en su conjunto.

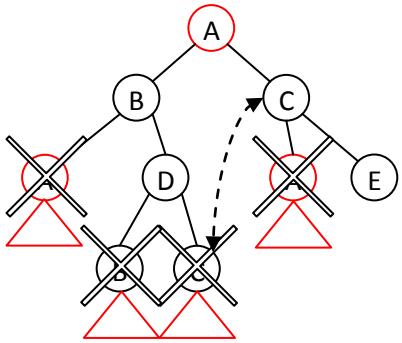


Ilustración 17 - Ejemplo de ambos casos anteriores juntos

La mejor forma de aplicar esta mejora es mediante las tablas de transposición, explicadas a continuación.

Tablas de transposición

Una transposición es aquella configuración de un tablero a la que se puede llegar con una combinación de movimientos distinta. Por ejemplo, en ajedrez podemos situar el caballo blanco derecho en g5 con la secuencia g1 → f3 → g5 o con la secuencia g1 → h3 → g5 (entre muchísimas otras posibles secuencias).

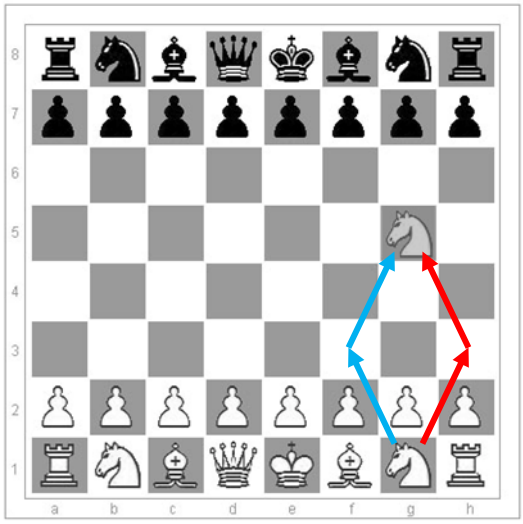


Ilustración 18 - Transposiciones en el ajedrez

El grave problema de las transposiciones es la expansión de nodos ya visitados en otras ramas de un árbol. En el siguiente ejemplo podemos ver dos transposiciones: los nodos C y E.

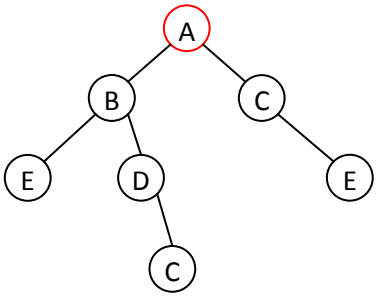


Ilustración 19 - Ejemplo de transposiciones

Evitando la expansión de nodos ya visitados se puede ahorrar muchos recursos, tanto de tiempo como de espacio en memoria (dependiendo del algoritmo de búsqueda utilizado). La forma de ahorrarse estos recursos son las tablas de transposición.

Estas tablas, dependiendo de su implementación, almacenan información sobre los nodos visitados y su coste heurístico (en caso de estar hablando de su uso sobre un algoritmo de búsqueda informada) con el fin de podar ramas del árbol en las que se puede predecir que se llegará al nodo final con un número de pasos mayor.

Una posible forma de identificar un nodo de manera unívoca y eficiente (en cuanto a consumo de espacio en memoria) es mediante una función hash que traduce la disposición de un nodo en un número o cadena alfanumérica (de igual tamaño para todos los nodos).

A continuación se muestra un ejemplo de nodo y su valor hash. Se trata de un tablero de 3x3 con dos piezas puestas en la posición (0, 2) y (2, 1) respectivamente:

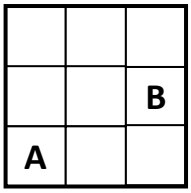


Ilustración 20 - Ejemplo de nodo y su valor hash

En este ejemplo se ha propuesto obtener la función hash de la siguiente fórmula:

$$hash = (posx(A) + 1 + 3 * posy(A)) * 10 + (posx(B) + 1 + 3 * posy(B))$$

y con ello se ha conseguido comprimir la información de disposición de un tablero de manera unívoca con 2 Bytes.

Una vez obtenida la función hash, cada vez que visitemos un nodo se guardará la información sobre el valor heurístico de dicho nodo, manteniendo la información actualizada y cada vez más precisa.

La fórmula con la que se ha de actualizar la información de la tabla de transposición es la siguiente:

$$tabla(hash) = \begin{cases} h(hoja), & si\ nodo\ hoja \\ h(hoja) + g(hoja) - g(nodo), & si\ nodo\ intermedio \end{cases}$$

Se puede implementar las tablas de transposición de tres formas distintas:

- **Listas estáticas**: Simplemente es un array en el que se guarda el valor heurístico en la posición del array designado por el valor hash. El acceso es constante pero el tamaño del problema está muy limitado ya que ocupa mucho espacio en memoria para un número de nodos grande.

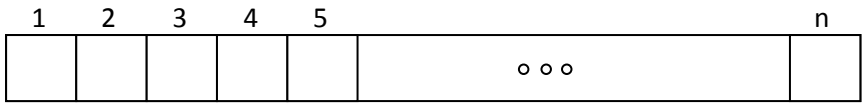


Ilustración 21 - Lista estática

- **Listas dinámicas**: Una lista enlazada en la que se añade un nuevo nodo cada vez que se tiene que asignar un valor heurístico a un valor hash no asignado anteriormente. Con este modelo se soluciona el problema del espacio de memoria consumido ya que únicamente aumenta si necesitamos añadir nuevos nodos. Sin embargo, el tiempo de acceso empeora considerablemente tanto en búsqueda como en inserción, concretamente consume tiempo lineal.

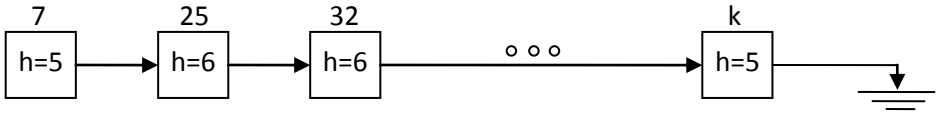


Ilustración 22 - Lista dinámica

- **Árboles de búsqueda**: En el caso de la realización de una tabla de transposición, el mejor árbol de búsqueda que se puede utilizar es un árbol de búsqueda binaria autobalanceable AVL. Este tipo de árbol fue ideado por dos matemáticos rusos llamados Adelson-Velskii y Landis (de ahí su nombre) y tiene una gran ventaja con respecto a los demás que es el acceso tanto para búsqueda como para inserción o borrado en tiempo logarítmico al mantener siempre una estructura arbórea balanceada. Por lo tanto, tiene la ventaja de las listas dinámicas (recursos de memoria) y, además, el tiempo de acceso es mucho mejor.

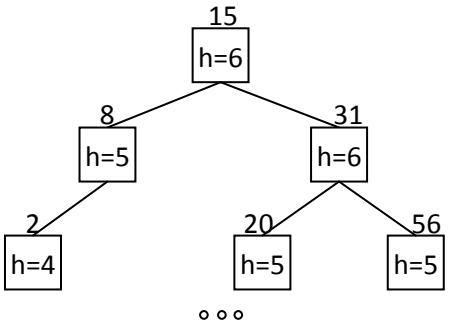


Ilustración 23 - Árbol AVL

A continuación se muestran dos tablas, comparando el coste temporal y espacial, de las tres estructuras de datos mostradas anteriormente para la implementación de las tablas de transposición:

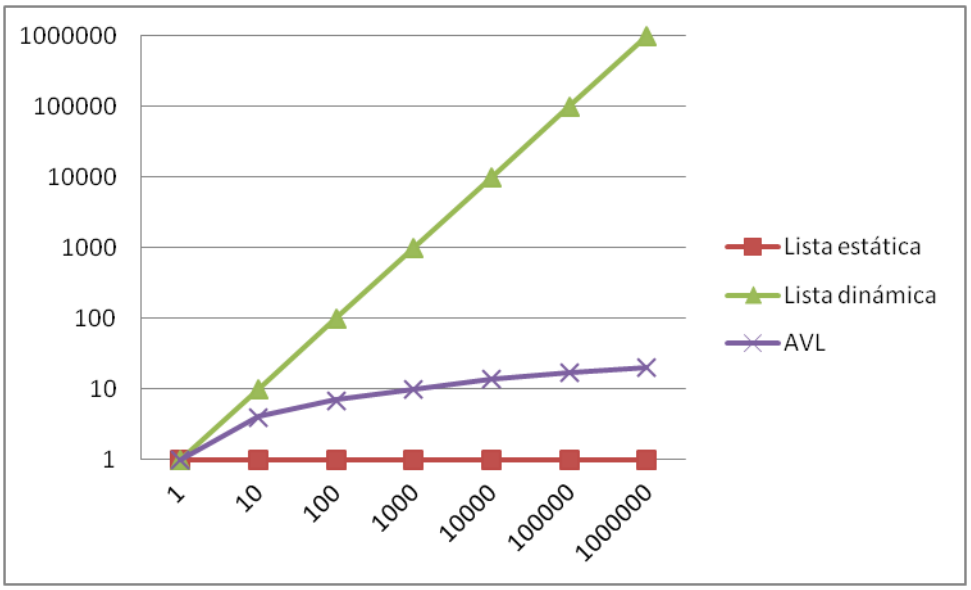


Ilustración 24 - Coste de tiempo de acceso

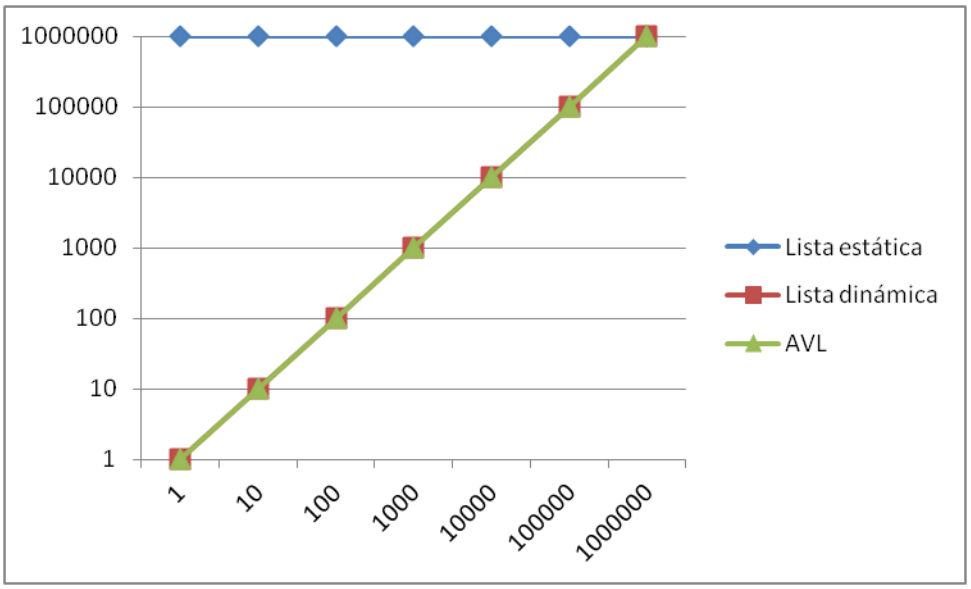


Ilustración 25 - Coste de recursos espaciales

Claramente, la mejor relación en cuanto a coste temporal y espacial es la del árbol AVL que, con el mismo coste espacial de una lista dinámica, se obtienen resultados de tiempo casi tan buenos como una lista estática.

Heurísticas

Las heurísticas[11] son unas funciones matemáticas que proporcionan información a los algoritmos de búsqueda informada sobre cada uno de los nodos para estimar el camino más corto desde cada uno de los nodos hasta el nodo meta.

Normalmente se implementa como una relajación del problema que se quiere resolver, es decir, se eliminan algunas restricciones. Por ejemplo, para el n-puzzle, una buena heurística sería la suma de la distancia Manhattan entre cada una de las piezas mal colocadas y su posición correcta.

NOTA: La distancia de Manhattan (bidimensional) es:

$$d_{Manhattan} = |x_i - x_f| + |y_i - y_f|$$

A continuación, un ejemplo de la distancia de Manhattan en un tablero de 3x3 en el que se quiere llegar desde (1, 1) hasta (3, 3):

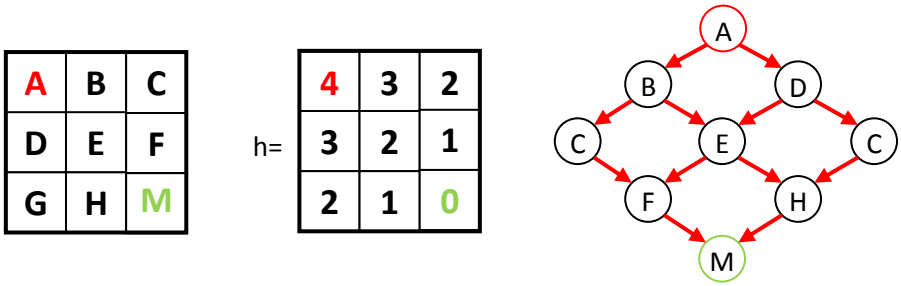


Ilustración 26- Izq) Tablero. Med) Coste heurístico de cada casilla Der) Tablero representado en grafo.

Sin embargo, si se quiere obtener un resultado óptimo con algoritmos de búsqueda informada que aseguren dicho resultado óptimo, es imprescindible seleccionar una heurística admisible. Una heurística admisible es aquella que no sobreestima, bajo ningún concepto, ninguno de los nodos. Sobreestimar un nodo implicaría la posibilidad de encontrar una solución no óptima, tal y como vemos en el siguiente ejemplo:

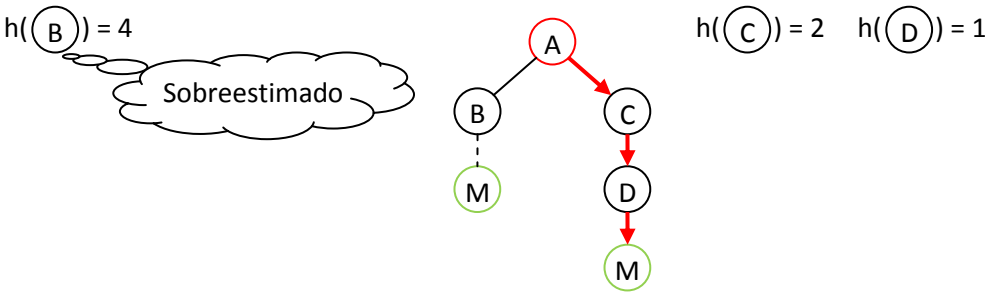


Ilustración 27 - Ejemplo de la no admisibilidad de una heurística

Se puede apreciar en este ejemplo que, dado que se considera que el nodo B tiene una heurística de 4, se expande primero el nodo C llegando hasta la meta con un coste mayor del que hubiese resultado con una heurística admisible.

Consecuencias

El efecto que producen las heurísticas en nuestro sistema es una mejora sustancial del tiempo de ejecución y del consumo de memoria, aunque todo depende de la calidad de la función heurística ya que ésta también tiene un coste de tiempo asociado.

En el caso de que la heurística no sea lo suficientemente buena y, a su vez, sea poco eficiente, ésta no será lo suficientemente buena como para ser rentable en su utilización.

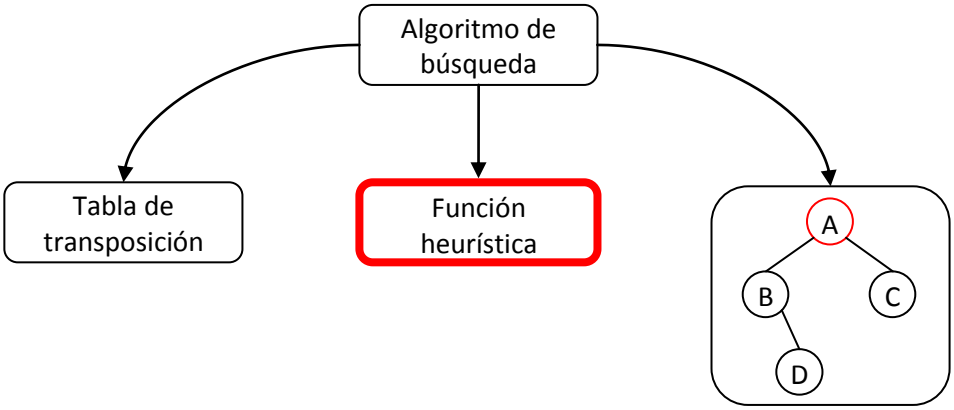


Ilustración 28 - Composición final del algoritmo IDA*

La importancia de que la heurística sea eficiente es muy elevada, ya que se va a aplicar a todos los nodos y por lo tanto el tiempo total invertido en el cálculo de heurísticas será:

$$t_{total\ heuristica} = t_{heuristica} * nodos$$

Capítulo 3

Objetivos

El objetivo principal de este proyecto es la creación de un sistema capaz de resolver, mediante algoritmos de inteligencia artificial y heurísticas, cualquier instancia resoluble del juego Rush Hour. Se ha de conseguir, además, la solución óptima para dichas instancias (el mínimo número de movimientos de los vehículos).

A continuación se muestra una lista con los objetivos:

- **Subsistemas:** El sistema se dividirá en dos partes importantes: La interfaz estará programada en Python mientras que el/los algoritmo(s) de búsqueda y heurística(s) estarán programados en C++ para conseguir una mayor velocidad en la ejecución.

- **Solución de instancias de 6x6:** El sistema será capaz de resolver instancias del juego Rush Hour de tamaño 6x6 con tamaño de vehículos estándar (2x1 y 3x1).

- **Selección de instancias por características:** El usuario indicará al sistema qué instancia quiere que se resuelva. Dichas instancias estarán ordenadas por dificultad.

- **Creación de nuevas instancias:** El usuario podrá crear una nueva instancia, mediante un editor, la cual podrá seleccionar posteriormente para que el sistema la resuelva. El sistema no realizará una comprobación de si la instancia es resoluble o no.

- **Edición de instancias:** El usuario podrá editar una instancia ya creada, mediante una interfaz gráfica, la cual podrá seleccionar posteriormente para que el sistema la resuelva. El sistema no realizará una comprobación de si la instancia es resoluble o no.

- **Salvado de instancias:** El usuario podrá guardar en un fichero la nueva instancia creada o editada para su posterior resolución.

- **Representación gráfica:** El sistema representará los movimientos que pertenecen a la solución de la instancia de forma gráfica.

- **Obtención de estadísticas:** El sistema mostrará las estadísticas de cada instancia (número de nodos generados, número de nodos expandidos, tiempo empleado, etc).

Capítulo 4

Desarrollo

Ésta es la sección en la que se describe el proceso mediante el cual se ha analizado, diseñado e implementado el agente que resolverá las instancias del juego Rush Hour. Este proceso se divide en varias partes:

- **Análisis del sistema**: en este apartado se estudia y descompone el problema en varias partes, definiendo los requisitos y casos de uso.

- **Toma de decisiones**: aquí se puede observar las distintas decisiones tomadas en cuanto a la forma en la que se ha modelado el problema, las plataformas utilizadas para la implementación, el algoritmo de búsqueda y heurísticas utilizadas así como mejoras del algoritmo de búsqueda.

- **Diseño del sistema**: en este apartado se muestra el modelo arquitectónico y diagrama de clases. Además, se explicará detalladamente el funcionamiento del algoritmo junto a la heurística.

- **Modelos 3D**: este breve apartado describe las técnicas que se han utilizado para realizar los modelos 3D con Blender.

Análisis del sistema

El problema se puede descomponer en varias subproblemas más sencillos, esto se puede comprobar en el apartado de requisitos del sistema y de casos de uso detallados a continuación.

Requisitos del sistema

En primer lugar se ha procedido a enumerar cada uno de los requisitos del sistema, distinguidos en 3 secciones:

- **Requisitos de interfaz**: son los requisitos de la interfaz de usuario en la que dicho usuario podrá interactuar con el sistema.

- **Requisitos de agente**: son los requisitos de la parte de inteligencia artificial que resolverá los tableros.

- **Requisitos de pruebas**: son los requisitos sobre las distintas pruebas que se han de realizar satisfactoriamente para que el sistema se considere completo.

Estos requisitos se pueden dividir, a su vez, en requisitos funcionales (si representan funcionalidad en el sistema) y requisitos no funcionales (en caso de que representen restricciones en el sistema).

Los requisitos se enumeran en tablas que contienen la siguiente información:

- **Identificador**: identifica el requisito unívocamente y consiste en un código de tipo de requisito (INT, AGE, PRU si es requisito de interfaz, de agente o de pruebas respectivamente), F o NF dependiendo de si se trata de un requisito funcional o no funcional y un código numérico compuesto de tres cifras, para distinguir requisitos del mismo tipo.

- **Nombre del requisito**: es un resumen del requisito en pocas palabras.

- **Descripción del requisito**: es una descripción detallada del requisito.

CÓDIGO-NO/FUNCIONAL-NÚMERO	
Nombre	Nombre del requisito
Descripción	Descripción del requisito.

Tabla 1 - Ejemplo de requisito

Requisitos de interfaz

INT-F-001	
Nombre	Visor de tableros
Descripción	Se podrá visualizar cada uno de los tableros almacenados.

Tabla 2 - INT-F-001

INT-F-002	
Nombre	Visor de resultados
Descripción	Se podrá visualizar los resultados obtenidos de cada uno de los tableros desde el visor de tableros o después de resolver un tablero en cuestión.

Tabla 3 - INT-F-002

INT-F-003	
Nombre	Creación de tableros
Descripción	Se podrá crear un nuevo tablero desde la interfaz.

Tabla 4 - INT-F-003

INT-F-004	
Nombre	Edición de tableros
Descripción	Se podrá crear un nuevo tablero a partir de otro tablero.

Tabla 5 - INT-F-004

INT-F-005	
Nombre	Visor de solución
Descripción	Se podrá observar cómo se resuelve el tablero mediante una animación.

Tabla 6 - INT-F-005

INT-NF-001	
Nombre	Interfaz sencilla
Descripción	La interfaz será sencilla de manejar incluso para usuarios principiantes.

Tabla 7 - INT-NF-001

Requisitos de agente

AGE-F-001	
Nombre	Resolver tableros
Descripción	El agente será capaz de resolver cualquier tablero con solución.

Tabla 8 - AGE-F-001

AGE-F-002	
Nombre	Detectar tableros sin solución
Descripción	El agente será capaz de detectar si un tablero no tiene solución.

Tabla 9 - AGE-F-002

AGE-F-003	
Nombre	Cálculo de estadísticas
Descripción	El agente devolverá estadísticas de tiempo, de movimientos y de nodos generados y expandidos.

Tabla 10 - AGE-F-003

AGE-NF-001	
Nombre	Eficiencia
Descripción	El agente resolverá las instancias de forma eficiente.

Tabla 11 - AGE-NF-001

Requisitos de pruebas

PRU-F-001	
Nombre	Nuevo tablero
Descripción	Se probará a crear un nuevo tablero y a guardarlo en un fichero.

Tabla 12 - PRU-F-001

PRU-F-002	
Nombre	Editar tablero
Descripción	Se probará a editar un tablero y a guardarlo en un fichero.

Tabla 13 - PRU-F-002

PRU-F-003	
Nombre	Resolver tablero
Descripción	Se probará a resolver cualquiera de los 40 tableros originales de Rush Hour.

Tabla 14 - PRU-F-003

Casos de uso

Los casos de uso nos permiten detallar la información sobre el sistema para aclarar la funcionalidad que este ofrecerá al usuario final.

Sólo hay un único actor que interactuará con la aplicación, se trata del usuario, el cuál podrá utilizar cualquier funcionalidad ofrecida por el software proporcionado.

A continuación se muestra un diagrama de casos de uso y la explicación de estos en tablas con los siguientes contenidos:

- **Identificador**: identifica el caso de uso con el código CU-XXX siendo XXX el número del caso de uso.
- **Nombre del caso de uso**: es un resumen del caso de uso en pocas palabras.
- **Descripción del caso de uso**: es una descripción detallada del caso de uso.
- **Actores**: son los actores involucrados en este caso de uso.
- **Precondiciones**: son las condiciones previas o estado del sistema antes de ejecutar la acción.
- **Postcondiciones**: es el estado del sistema después de ejecutar la acción.
- **Secuencia**: es la secuencia de tareas que se realizan para finalizar acción.
- **Secuencia alternativa**: es la secuencia que se realiza si se produce un error.

CU-NÚMERO	
Nombre	Nombre del caso de uso.
Descripción	Descripción del caso de uso.
Actores	Los actores implicados.
Precondiciones	El estado en el que hay que encontrarse para poder realizar la acción.
Postcondiciones	El estado en el que se queda el sistema después de realizar la acción.
Secuencia	La secuencia de tareas que realiza el actor/sistema para finalizar la acción.
Secuencia alternativa	La secuencia alternativa de tareas que realiza el actor/sistema para finalizar la acción. Se produce típicamente cuando hay un error.

Tabla 15 - Ejemplo de caso de uso

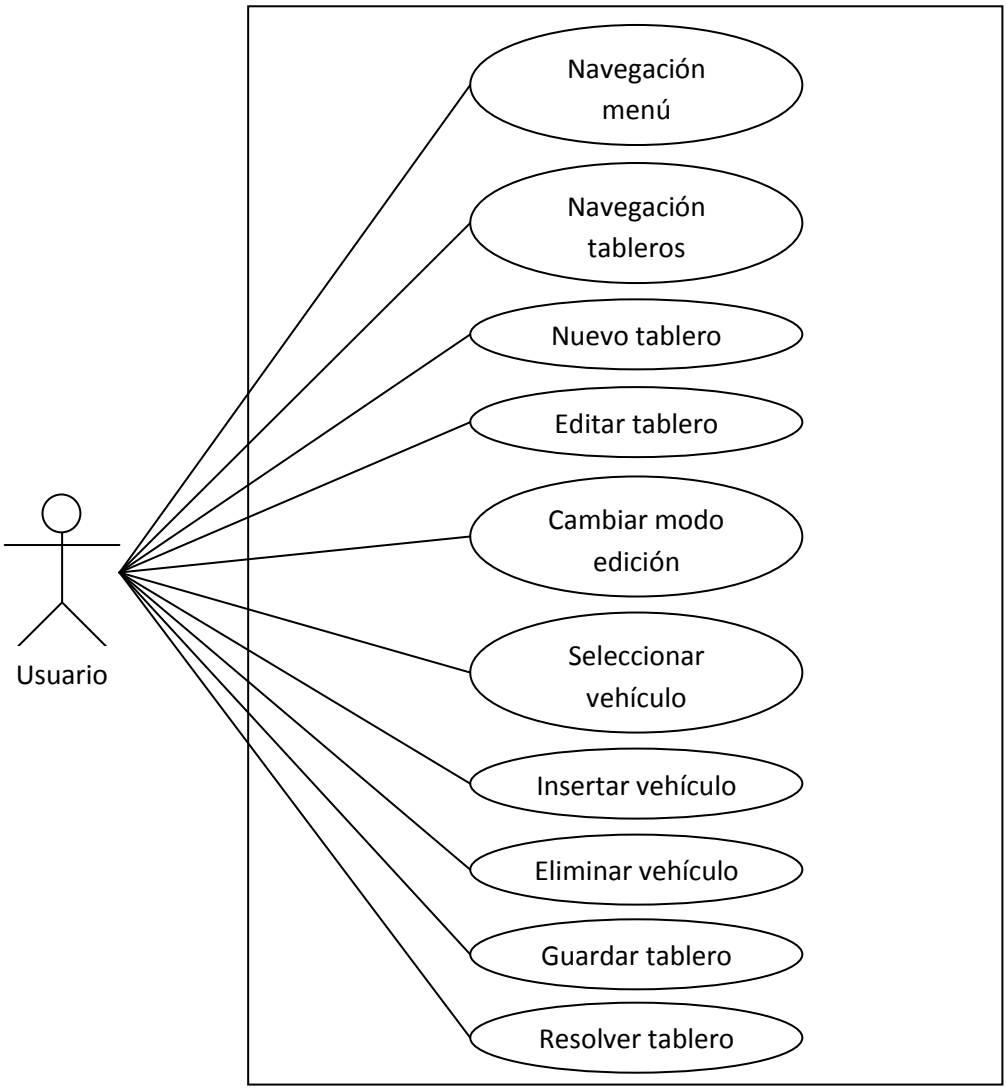


Ilustración 29 - Diagrama de casos de uso

CU-001	
Nombre	Navegación menú
Descripción	El usuario navega a través del menú, tanto hacia adelante como hacia atrás.
Actores	Usuario
Precondiciones	Situarse en cualquier lugar del menú.
Postcondiciones	Situarse en el lugar destino del menú.
Secuencia	1- El usuario hace *click* en la entrada del menú hacia donde quiere navegar. 2- El sistema obtiene el destino hacia donde se está navegando. 3- El sistema averigua si hay que ir hacia adelante o hacia atrás. 4- El sistema muestra una animación de cambio de menú. 5- Se muestra el menú de destino.
Secuencia alternativa	1- El usuario hace *click* en "Salir". 2- El sistema termina.

Tabla 16 - CU-001

CU-002	
Nombre	Navegación tableros
Descripción	El usuario observa los distintos tableros guardados anteriormente y sus estadísticas (en caso de haber sido resueltos previamente).
Actores	Usuario
Precondiciones	Situarse dentro del submenú de cualquier nivel de dificultad.
Postcondiciones	Situarse dentro del submenú de cualquier nivel de dificultad cambiando el tablero observado.
Secuencia	1- El usuario hace *click* la flecha de navegación de tableros (izquierda o derecha). 2- El sistema comprueba si hay más tableros en la dirección pulsada. 3- Si hay, el sistema realiza una animación de cambio de tablero. 4- Se muestra el tablero siguiente.
Secuencia alternativa	3- Si no hay, el sistema no hace nada. 4- El sistema termina.

Tabla 17 - CU-002

CU-003	
Nombre	Nuevo tablero
Descripción	El usuario edita un nuevo tablero desde cero.
Actores	Usuario
Precondiciones	Situarse dentro del submenú de cualquier nivel de dificultad dentro del submenú editor.
Postcondiciones	Editor abierto con un tablero vacío.
Secuencia	1- El usuario hace *click* en aceptar en el tablero “Nuevo”. 2- El sistema cambia al editor. 3- El sistema muestra un tablero vacío y habilita la disposición de todos los vehículos.
Secuencia alternativa	Ninguna

Tabla 18 - CU-003

CU-004	
Nombre	Editar tablero
Descripción	El usuario modifica un tablero previamente guardado.
Actores	Usuario
Precondiciones	Situarse dentro del submenú de cualquier nivel de dificultad dentro del submenú editor.
Postcondiciones	Editor abierto con el tablero a editar.
Secuencia	1- El usuario hace *click* en cualquier tablero. 2- El sistema cambia al editor. 3- El sistema muestra el tablero y habilita la disposición de los vehículos no utilizados previamente en el tablero.
Secuencia alternativa	Ninguna

Tabla 19 - CU-004

CU-005	
Nombre	Cambiar modo edición
Descripción	El usuario cambia a disposición del vehículo en horizontal, vertical o a modo de borrado.
Actores	Usuario
Precondiciones	Situarse en el editor.
Postcondiciones	Cambio del modo de edición.
Secuencia	1- El usuario hace *click* en uno de los 3 modos (horizontal, vertical o borrado). 2- El sistema cambia el modo. 3- Se muestra por pantalla el modo actual.
Secuencia alternativa	Ninguna

Tabla 20 - CU-005

CU-006	
Nombre	Seleccionar vehículo
Descripción	Selecciona un vehículo para utilizarlo posteriormente.
Actores	Usuario
Precondiciones	Situarse en el editor.
Postcondiciones	Cambia el vehículo seleccionado.
Secuencia	1- El usuario hace *click* en uno de los vehículos de la parte derecha de la pantalla. 2- El sistema cambia el vehículo seleccionado. 3- Se muestra por pantalla el vehículo seleccionado.
Secuencia alternativa	Ninguna

Tabla 21 - CU-006

CU-007	
Nombre	Insertar vehículo
Descripción	Inserta el vehículo seleccionado en el tablero, en la posición y orientación seleccionadas.
Actores	Usuario
Precondiciones	Situarse en el editor y no estar activo el modo de borrado.
Postcondiciones	Vehículo insertado y actualizada la disposición.
Secuencia	1- El usuario hace *click* en una posición del tablero. 2- El sistema comprueba si se puede insertar el vehículo en el tablero. a. El vehículo está disponible. b. El vehículo no se sale del tablero. c. El vehículo no cae en una casilla ocupada. d. Además, si se trata del coche principal, el vehículo tiene que estar en posición horizontal y en $y = 3$. 3- El sistema inserta el vehículo seleccionado en la posición y orientación indicadas. 4- El sistema disminuye en 1 la disposición del vehículo seleccionado. 5- Se actualiza el tablero y la disposición del vehículo insertado.
Secuencia alternativa	3- El sistema no puede insertar el vehículo ya que no cumple alguna de las condiciones de 2).

Tabla 22 - CU-007

CU-008	
Nombre	Eliminar vehículo
Descripción	Elimina el vehículo situado en la posición indicada.
Actores	Usuario
Precondiciones	Situarse en el editor y estar activo el modo de borrado.
Postcondiciones	Vehículo eliminado y actualizada la disposición.
Secuencia	1- El usuario hace *click* en una posición del tablero. 2- El sistema comprueba si hay un vehículo en esa posición. 3- El sistema elimina el vehículo que esté ocupando dicha posición. 4- El sistema aumenta en 1 la disposición del vehículo eliminado. 5- Se actualiza el tablero y la disposición del vehículo eliminado.
Secuencia alternativa	3- El sistema no puede borrar el vehículo ya que no hay ninguno en la posición indicada.

Tabla 23 - CU-008

CU-009	
Nombre	Guardar tablero
Descripción	Se guarda el tablero creado/editado en un fichero.
Actores	Usuario
Precondiciones	Situarse en el editor.
Postcondiciones	Tablero guardado.
Secuencia	1- El usuario introduce un nombre y hace *click* en el botón “Guardar”. 2- El sistema comprueba si el tablero es correcto (al menos está el vehículo principal) y si el nombre está libre. 3- Si es así, se guarda el tablero en un fichero con el nombre indicado. 4- Se muestra un mensaje de confirmación.
Secuencia alternativa	3- Si no es así, se muestra un mensaje de error.

Tabla 24 - CU-009

CU-010	
Nombre	Resolver tablero
Descripción	Se resuelve el tablero seleccionado, se muestra la solución óptima y las estadísticas obtenidas.
Actores	Usuario
Precondiciones	Situarse dentro del submenú de cualquier nivel de dificultad dentro del submenú resolver.
Postcondiciones	Tablero resuelto y solución y estadísticas mostradas.
Secuencia	1- El usuario hace *click* en cualquier tablero. 2- El agente es llamado ofreciéndole las posiciones de los vehículos. 3- El agente resuelve el tablero (o no si éste no tiene solución). 4- El agente devuelve al sistema los movimientos realizados para resolver el tablero así como las estadísticas de nodos generados, nodos expandidos, tiempo y movimientos realizados 5- El sistema muestra una animación de cómo se resuelve el tablero. 6- El sistema muestra las estadísticas obtenidas.
Secuencia alternativa	5- Si el tablero no tiene solución, muestra directamente las estadísticas.

Tabla 25 - CU-010

Toma de decisiones

Desde el principio se ha tenido en cuenta una clara diferenciación de dos subsistemas dentro de este proyecto. El subsistema interfaz, el cuál iba a ser la parte en la que el usuario puede interactuar con el sistema; y el subsistema agente, el cuál iba a ser el encargado de resolver cualquier instancia de Rush Hour.

El subsistema agente se puede dividir en 3 partes: el algoritmo de búsqueda, la heurística y la tabla de transposición.

En esta sección del documento se muestra una breve explicación de las decisiones tomadas sobre diversos aspectos de estos dos subsistemas.

Interfaz

La interfaz, al ser la parte en la que el usuario interactúa con el sistema, tenía que ser algo notable a la vista, por lo que se decidió realizar una interfaz con modelos 3D y animaciones que representasen la resolución de los tableros.

Estos modelos 3D han sido realizados en un software de carácter libre muy conocido y utilizado en la actualidad, que además se está mejorando constantemente para ofrecer al usuario todas las características (e incluso más) que cualquier software de pago conocido (como 3DS Max). El software del que hablamos es Blender[12].

Ilustración 30 - Logo Blender

Para la edición de texturas y otras imágenes, se ha utilizado también software libre, concretamente el conocido GIMP[13].

Ilustración 31 - Logo GIMP

En cuanto a la programación de la interfaz, se ha realizado en el lenguaje de programación Python debido a su sencillo uso, su gran amplitud de librerías estándar. La versión utilizada es la que viene incluída en el paquete de Panda3D.

Ilustración 32 - Logo Python

Como motor gráfico se ha utilizado las librerías de panda3D[14], un potente motor que se puede usar tanto en Python como en C++. La decisión de utilizar Python en la interfaz ha venido dada por la facilidad de aprendizaje de este lenguaje y aún más para un motor gráfico que, personalmente, nunca había utilizado.

Ilustración 33 - Logo Panda3D

Además, para la programación de la interfaz se ha utilizado el entorno de edición de texto de software libre, gedit[15].

Ilustración 34 - Logo gedit

Agente

En cuanto al agente, se refiere a la parte de inteligencia artificial encargada de resolver cada uno de los tableros, es decir, el algoritmo de búsqueda, la heurística y la tabla de transposiciones.

Todo esto se ha programado en el lenguaje de programación C++. La decisión ha sido tomada en base a la diferencia de eficiencia entre el lenguaje Python y C++ ya que este último es mucho más rápido al tratarse de un lenguaje compilado y no interpretado.

Ilustración 35 - Logo C++

Mientras que el editor seleccionado para implementar la funcionalidad descrita ha sido Dev-C++, que incluye las herramientas necesarias para compilar y ejecutar el código escrito.

Ilustración 36 - Logo Dev-C++

El mayor de los problemas a la hora de utilizar dos lenguajes de programación distintos es que hay que interconectarlos de algún modo, de tal forma que uno de los bloques llame al otro y éste le devuelva unos resultados. En el caso de que haya que pasar, además, parámetros de entrada, el asunto se complica extremadamente por lo que se han tomado las siguientes decisiones al respecto:

Se ha utilizado una librería para interconectar ambos códigos, llamada SWIG[16] (Simplified Wrapper and Interface Generator).

Ilustración 37 - Logo SWIG

Además, los parámetros de entrada y resultados se han pasado por un fichero externo llamado shared.rh para facilitar la conexión entre ambos módulos.

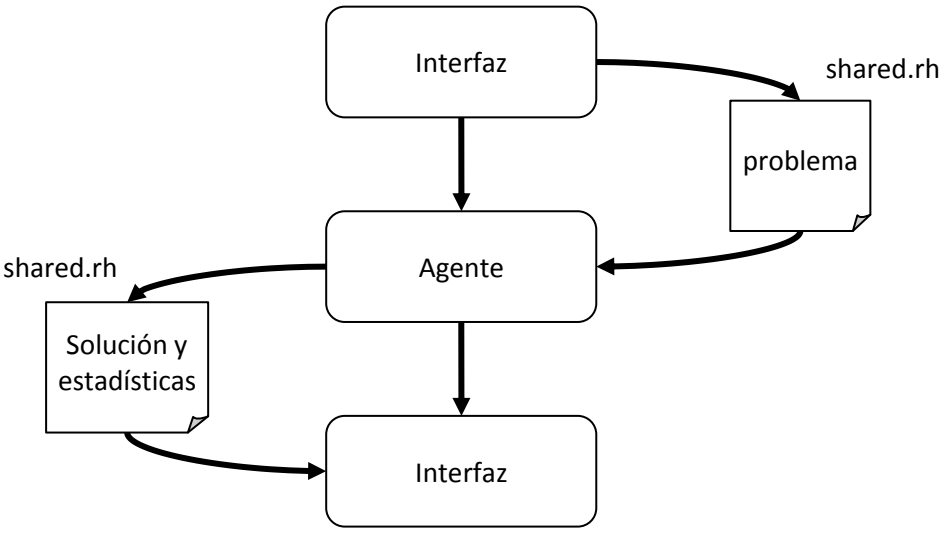


Ilustración 38 - Estructura básica de funcionamiento

En cuanto a decisiones de codificación, se han tomado las siguientes con respecto a la parte de inteligencia artificial

Algoritmo de búsqueda

El algoritmo de búsqueda utilizado ha sido el IDA* debido a su gran rendimiento, pero sobre todo a la capacidad de resolver un problema independientemente de lo grande que sea por su consumo constante de recursos de espacio.

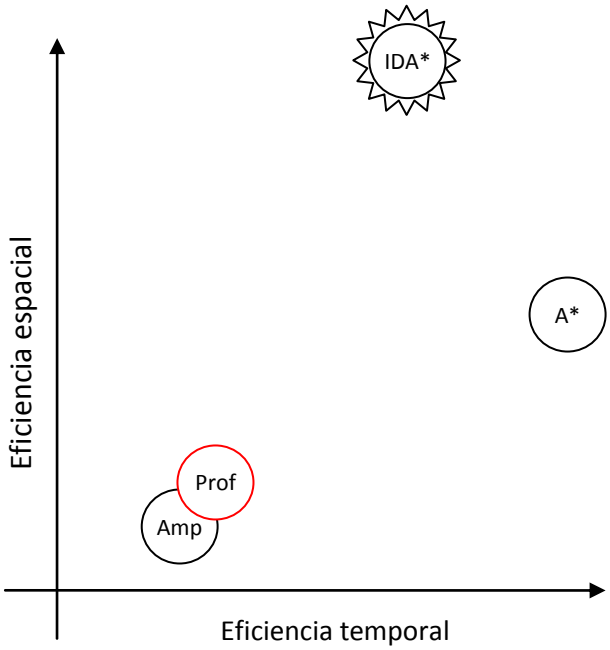


Ilustración 39 - Eficiencia cualitativade algunos algoritmos de búsqueda

Heurísticas

Se ha decidido utilizar dos heurísticas sencillas para comprobar la diferencia de eficiencia en cada una de ellas. Estas heurísticas están explicadas en el capítulo 5 de resultados:

- **Heurística 1**: Distancia del coche principal a la meta.

- **Heurística 2**: Distancia del coche principal a la meta más el número de vehículos que interfieren.

Tabla de transposición

En cuanto a la forma de evitar la expansión de nodos repetidos, se ha utilizado una tabla de transposición. Como se describe en el capítulo 2, hay varias formas de implementar estas tablas.

La mejor de ellas se trata de implementar un árbol binario de búsqueda autobalanceable AVL, ya que se consigue evitar el problema de los nodos duplicados con un tiempo de acceso muy bueno, concretamente se trata de un tiempo de acceso logarítmico tanto para la inserción como para la búsqueda. Al ser un árbol autobalanceable, siempre se va a asegurar el mismo tiempo de acceso.

Una mejora ha sido la de dividir la tabla de transposición en dos posibilidades:

- **Tabla de transposición en una lista estática**: Para 8 vehículos o menos, ya que ocupará 390625 posiciones, o lo que es lo mismo, tan sólo unos 382KB de memoria.

- **Tabla de transposición en el árbol AVL**: Para más de 8 vehículos.

Con esto se mejora radicalmente el tiempo de acceso para los tableros más sencillos, concretamente el tiempo pasa a ser constante.

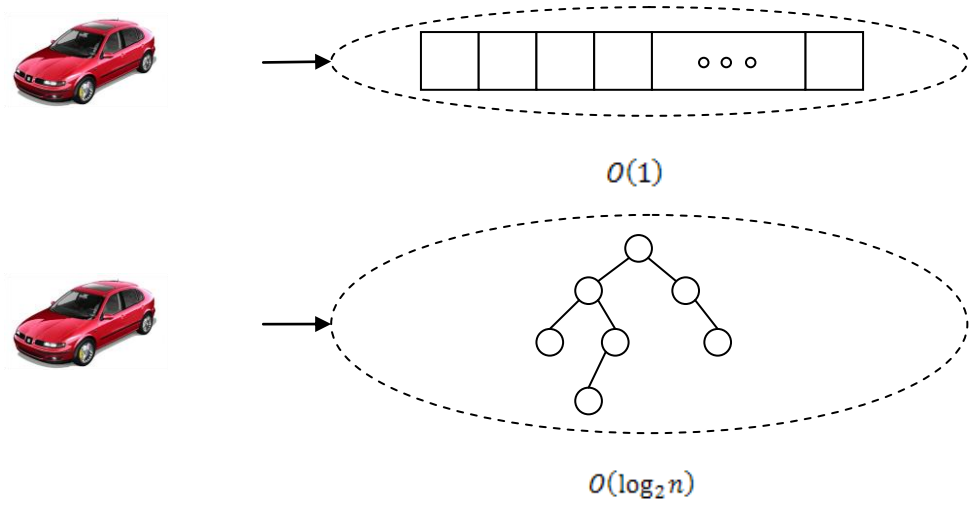


Ilustración 40 - Estructura de datos para las tablas de transposición

Conclusión

Para terminar con este apartado, a continuación se muestra un diagrama con el funcionamiento a grandes rasgos del sistema:

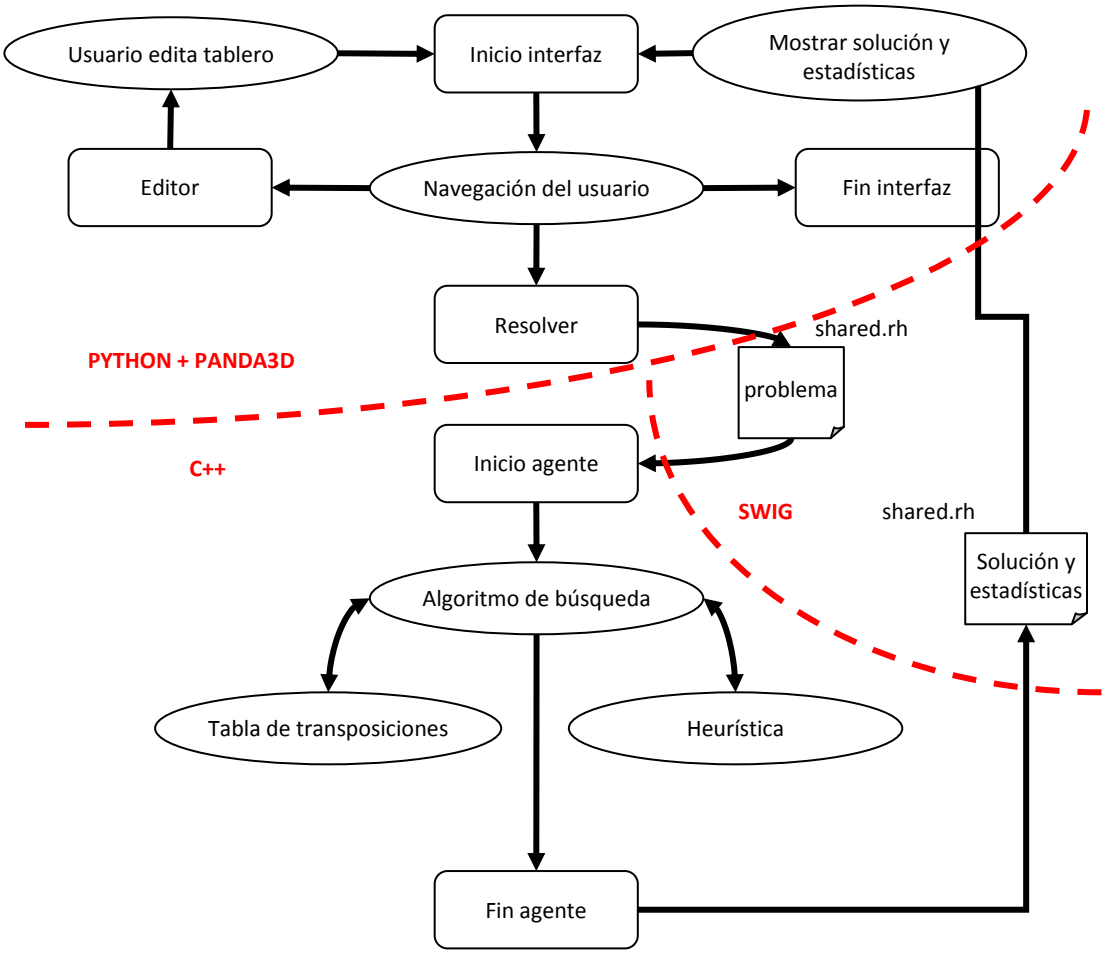


Ilustración 41 - Funcionamiento a grandes rasgos del sistema

Diseño

Con los diagramas de clases UML se puede definir de forma explícita la estructura que tiene el sistema y la relación que hay entre cada uno de los módulos y clases.

El diseño se ha dividido en dos grandes apartados:

- **Interfaz gráfica**: Es la parte escrita en lenguaje Python y en la que el usuario interactúa con el sistema.

- **Agente**: Es la parte escrita en el lenguaje de programación C++. Ésta es la parte más importante del proyecto ya que se encarga de resolver las instancias de los tableros.

Interfaz gráfica

La parte de la interfaz gráfica viene definida por los siguientes diagramas de clases, dividiendo el diagrama principal en varias partes para poder observarlo con más claridad y explicarlo por separado.

Vehiculo

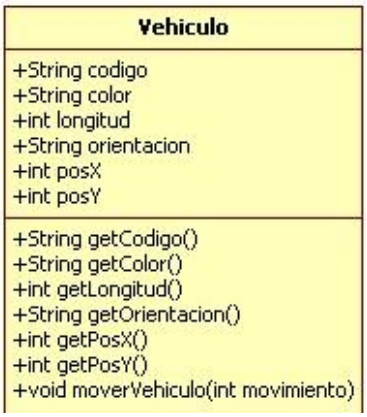


Ilustración 42 - Clase Vehiculo

- **Vehiculo**: Clase que guarda la información relativa a un vehículo, tales como la posición, longitud, etc.

FuncionesArbolRender

- **FuncionesArbolRender**: Esta es la clase que se encarga de realizar las gestiones para introducir o eliminar los nodos y "reparentarlos" con el nodo render del motor gráfico para poder representarlos visualmente.

También realiza otras gestiones como el cálculo de choques de vehículos en el editor para no permitir incluir un vehículo en una casilla ya ocupada.

- **Nodo**: Para poder manejar todos los nodos de manera sencilla es necesario realizar una estructura arbórea independientemente de la estructura en árbol que utiliza el motor gráfico Panda3D para la renderización de los elementos 2D/3D. Esta estructura se compone de objetos de la clase Nodo mostrada en el gráfico de la página siguiente y cada nodo puede tener entre 0 y N nodos hijos 3D y entre 0 y N nodos hijos 2D. De esta forma se puede ir recorriendo el árbol para obtener las instancias de Nodos renderizables y así poder realizar operaciones básicas (traslación, escalado y rotación) sobre dichas instancias.

- **Vehiculo**: Explicado en el diagrama anterior. Es necesario tener esta clase en este módulo ya que tanto la clase Nodo como FuncionesArbolRender hacen uso de instancias de Vehiculo.

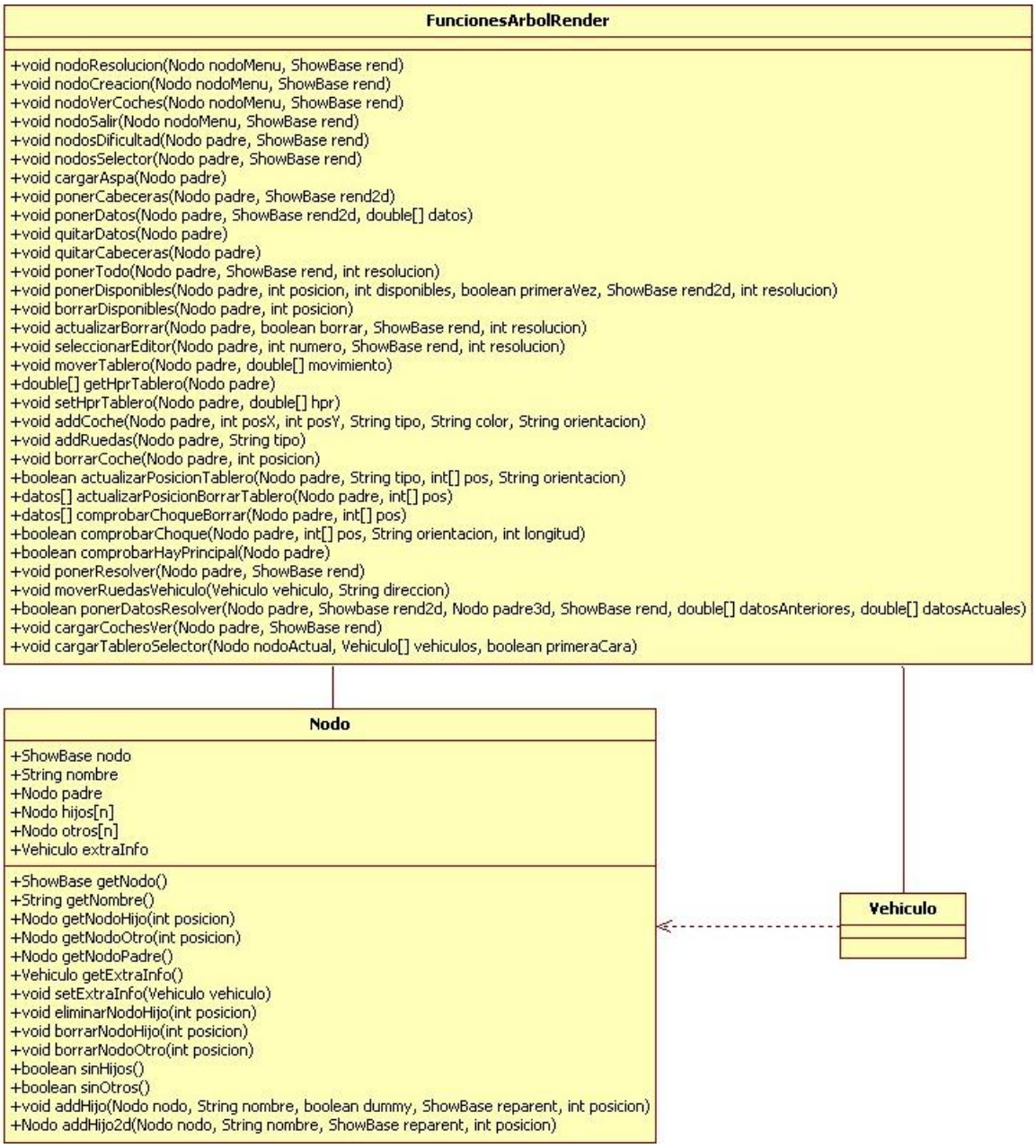


Ilustración 43 - Clase FuncionesArbolRender y Nodo

Funciones

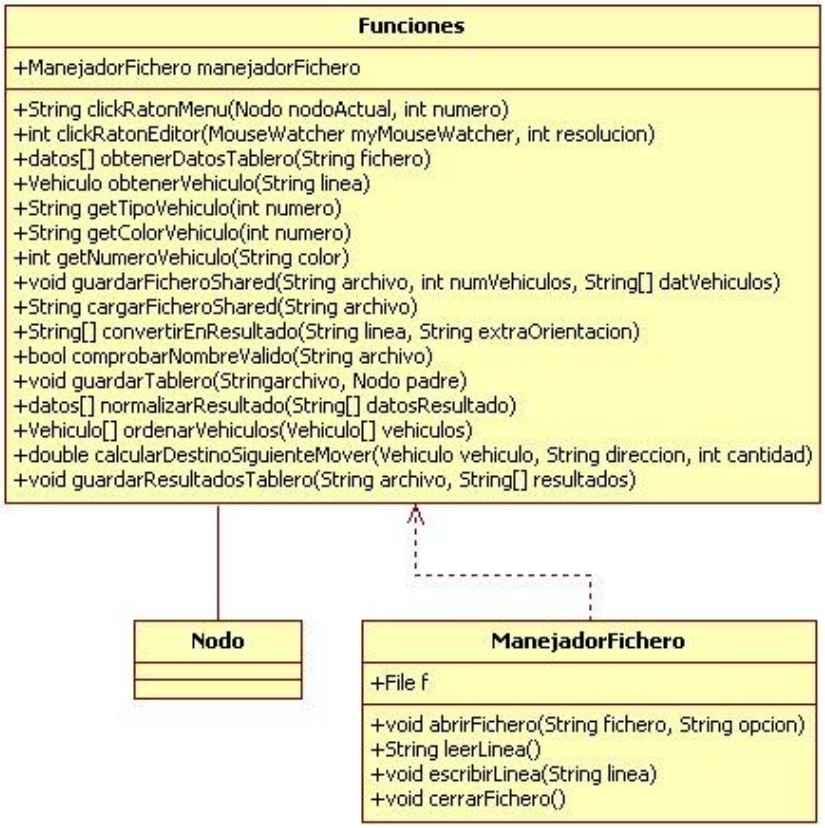


Ilustración 44 - Clase Funciones y ManejadorFichero

- **Funciones**: Esta clase está construida básicamente para liberar gran parte de la carga de funciones de otras clases como Interfaz. En ella se declaran funciones para manejo de ficheros (obtención y grabado de datos) y para la toma de decisiones al hacer "click" con el ratón.
- **ManejadorFichero**: Esta clase es la encargada del acceso directo a los ficheros, tanto para la obtención como para la escritura, línea a línea, de datos.
- **Nodo**: Esta clase está explicada en la sección anterior. La clase Funciones requiere de nodos, por lo tanto hace uso de ésta.

EventosTask

- **EventosTask**: Es una clase encargada de realizar los cálculos para saber qué evento se producirá por estar el ratón en cierta parte de la pantalla. Utilizado en los menús y el editor para obtener una respuesta visual que facilite la interactuación del usuario con el sistema.

- **Nodo**: Esta clase está explicada en el apartado anterior. EventosTask hace uso de esta clase para aplicar la respuesta visual a una instancia de la clase Nodo y así ser visible por el usuario.

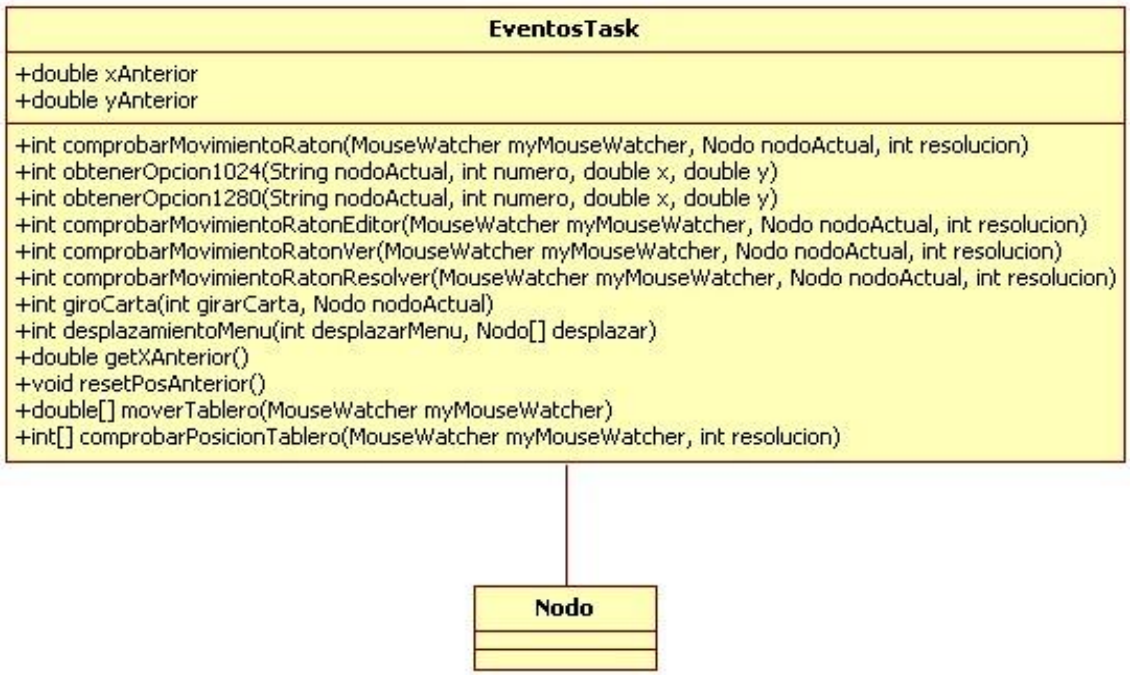


Ilustración 45 - Clase EventosTask

Interfaz

- **Interfaz**: Es la clase más importante de todo el módulo de interfaz gráfica. Ella se encarga de realizar las tareas de las animaciones y de llamar a las clases más importantes para realizar operaciones (por ejemplo, a FuncionesArbolRender). También recoge los eventos de ratón realizados (tanto pulsar como levantar) para realizar las operaciones pertinentes dependiendo del estado en el que se encuentre la máquina de estados.

Las tareas Task son unas operaciones que se realizan continuamente y que, dada la naturaleza del motor gráfico, su frecuencia de realización depende del número de cuadros por segundo que pueda aportar la tarjeta gráfica. Estas operaciones realizan animaciones y su estructura de ejecución es la siguiente:

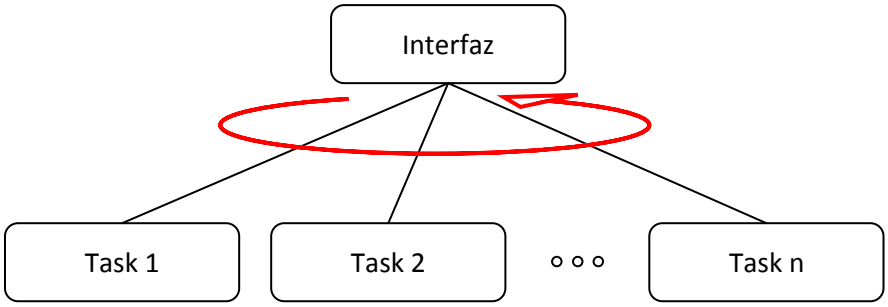


Ilustración 46 - Estructura de ejecución de las tareas

- **FuncionesArbolRender**: Interfaz tiene una instancia de esta clase para realizar llamadas sobre ella y así "dirigir" la colocación de los elementos de la interfaz.

- **Nodo**: La interfaz también posee instancias de la clase Nodo. Las más importantes son aquellas que identifican cada uno de los apartados del menú, que son los nodos raíz.

- **Vehículo**: esta clase es utilizada para guardar información sobre los vehículos a cargar, tanto a la hora de editar un tablero como a la hora de resolverlo.

- **EventosTask**: La Interfaz tiene una instancia de esta clase para realizar las comprobaciones explicadas en el apartado anterior.

- **Funciones**: La interfaz también posee una instancia de la clase Funciones utilizada para delegar parte de las funciones a esta clase.

- **ManejadorFichero**: Esta clase está explicada anteriormente.

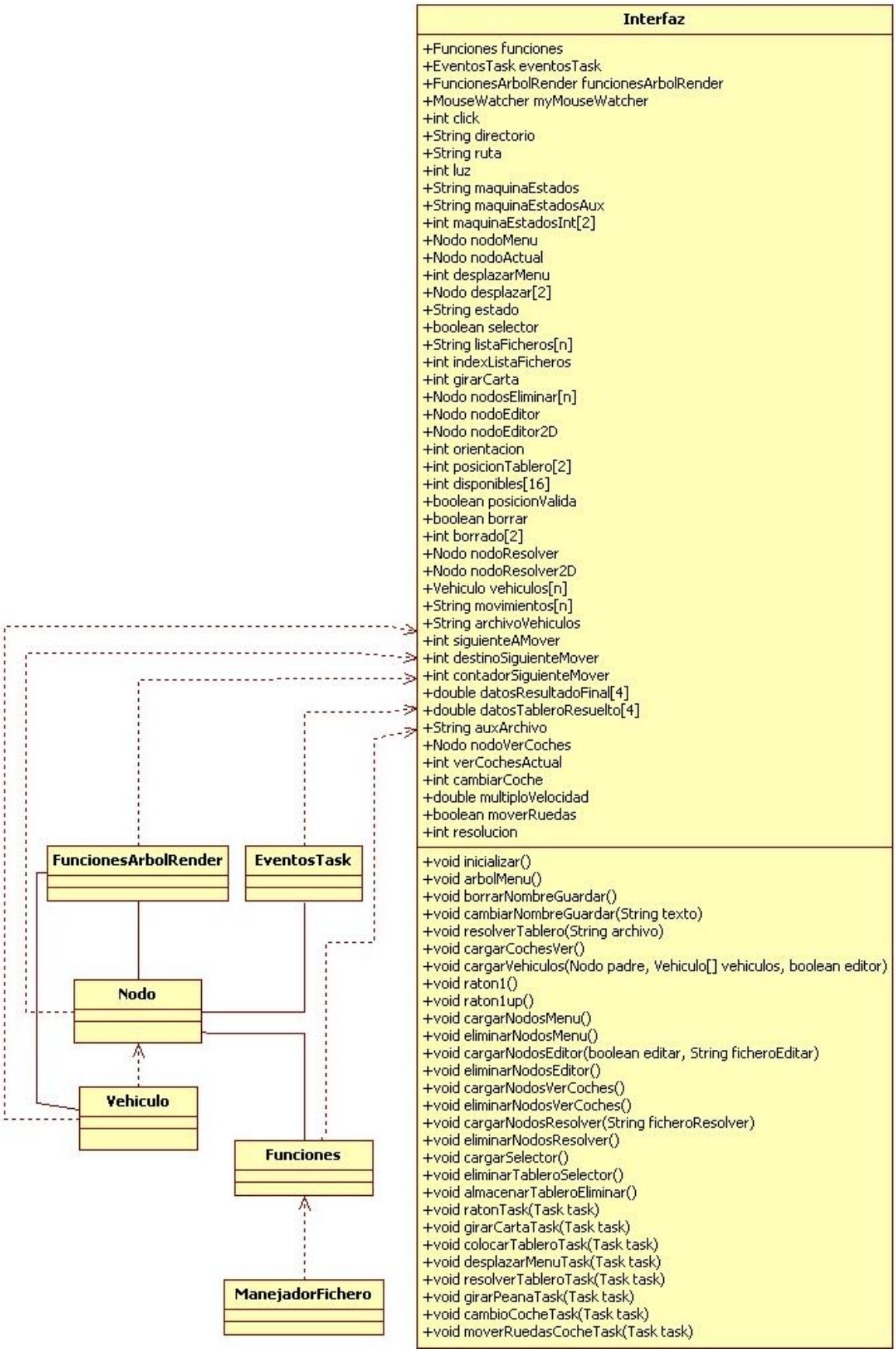


Ilustración 47 - Clase Interfaz

Diagrama completo

En el código se ha añadido una última clase llamada RushHourPFC que es la clase principal desde la que se ejecuta el resto del código. El diagrama completo, por lo tanto, es el siguiente:

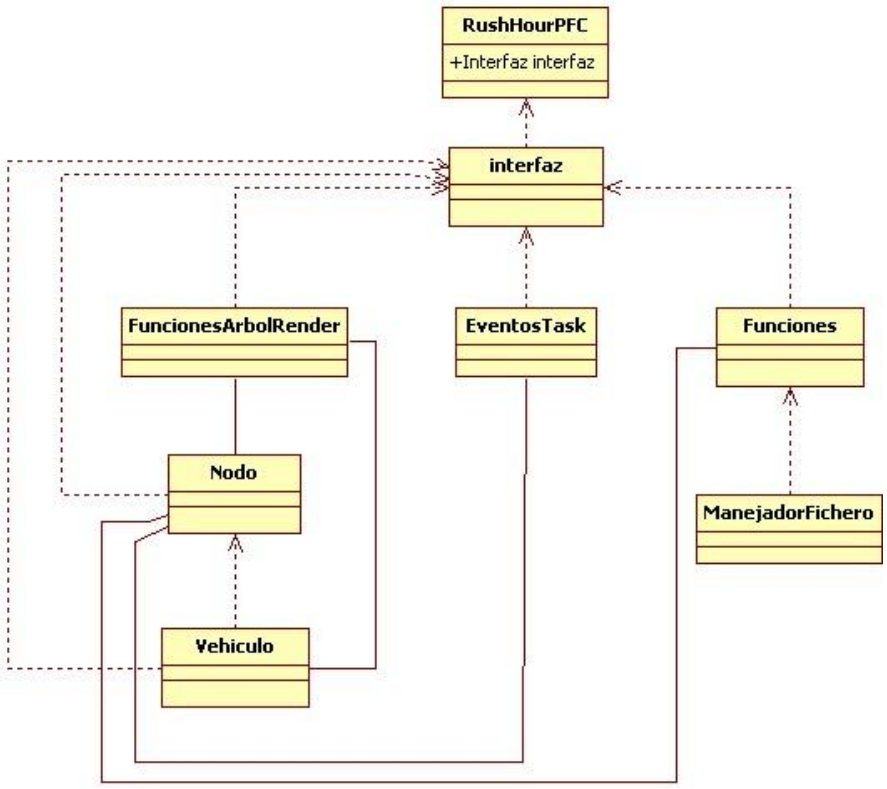


Ilustración 48 - Diagrama completo para el módulo de interfaz

Agente

El agente viene definido por los siguientes diagramas de clases, dividiendo el diagrama principal en varias partes para poder observarlo con más claridad y explicarlo por separado.

Tabla de transposiciones

La tabla de transposiciones es uno de los apartados más importantes que se ha realizado en este proyecto. Está dividida en dos clases:

- **TablaHash**: Esta clase es la que guarda el nodo raíz de la tabla de transposiciones y mantiene el árbol AVL correctamente balanceado después de insertarse cada uno de los nodos de la TablaHash.

- **HashNodo**: El árbol AVL se compone de instancias de estos nodos, que guardan información sobre el nodo padre, el hijo izquierdo y el hijo derecho. Además, se almacenan otros datos como el balance, el número hash del nodo que se ha expandido/generado y el valor heurístico del mismo.

Ilustración 49 - Clase TablaHash y HashNodo

Heuristica

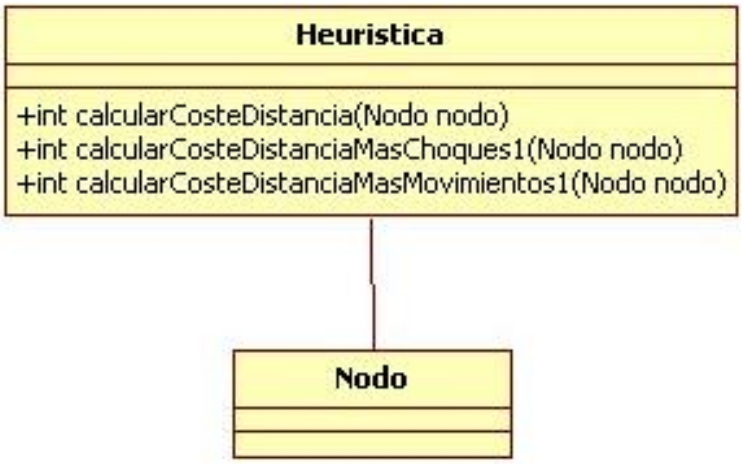


Ilustración 50 - Clase Heuristica

- **Heuristica**: Esta clase es la encargada de realizar los cálculos para el valor heurístico de un nodo determinado. Ese nodo se le pasa por parámetro a una de las tres funciones heurísticas posibles a utilizar.
- **Nodo**: La clase Heuristica utiliza instancias de la clase nodo. Esta clase está explicada con detalle en el siguiente apartado.

IDA

El diagrama siguiente comprende al algoritmo de búsqueda IDA* incluyendo la heurística y la tabla de transposición:

- **IDA**: Esta clase es la que realiza las operaciones necesarias para resolver cualquier instancia que obtenga del fichero shared.rh. Implementa el algoritmo de búsqueda IDA* explicado en el capítulo 2, guardando únicamente un nodo en memoria para ahorrar el máximo de recursos espaciales.

- **Nodo**: Es la clase que proporciona a IDA la capacidad de guardar información sobre la disposición de los vehículos y el número hash del tablero.

La información guardada en dicho nodo está duplicada de tal forma que mejora el tiempo de resolución. Esto se debe a que se guarda tanto la información de cada uno de los vehículos (posición, orientación y tamaño) como la disposición del tablero completo (muestra qué casillas están ocupadas por qué vehículo).

- **Heuristica**: Esta clase ya ha sido explicada en la sección anterior. La clase IDA tiene una instancia de Heuristica para poder obtener el valor heurístico de un nodo en determinadas partes del algoritmo.

- **TablaHash**: La clase IDA tiene también una instancia de la TablaHash que, en realidad, es la tabla de transposiciones. Con esta instancia, en determinadas zonas del algoritmo de búsqueda se obtienen valores heurísticos ya obtenidos o se insertan nodos para la tabla de transposiciones dependiendo del valor hash.

- **HashNodo**: Esta clase ya ha sido explicada anteriormente.

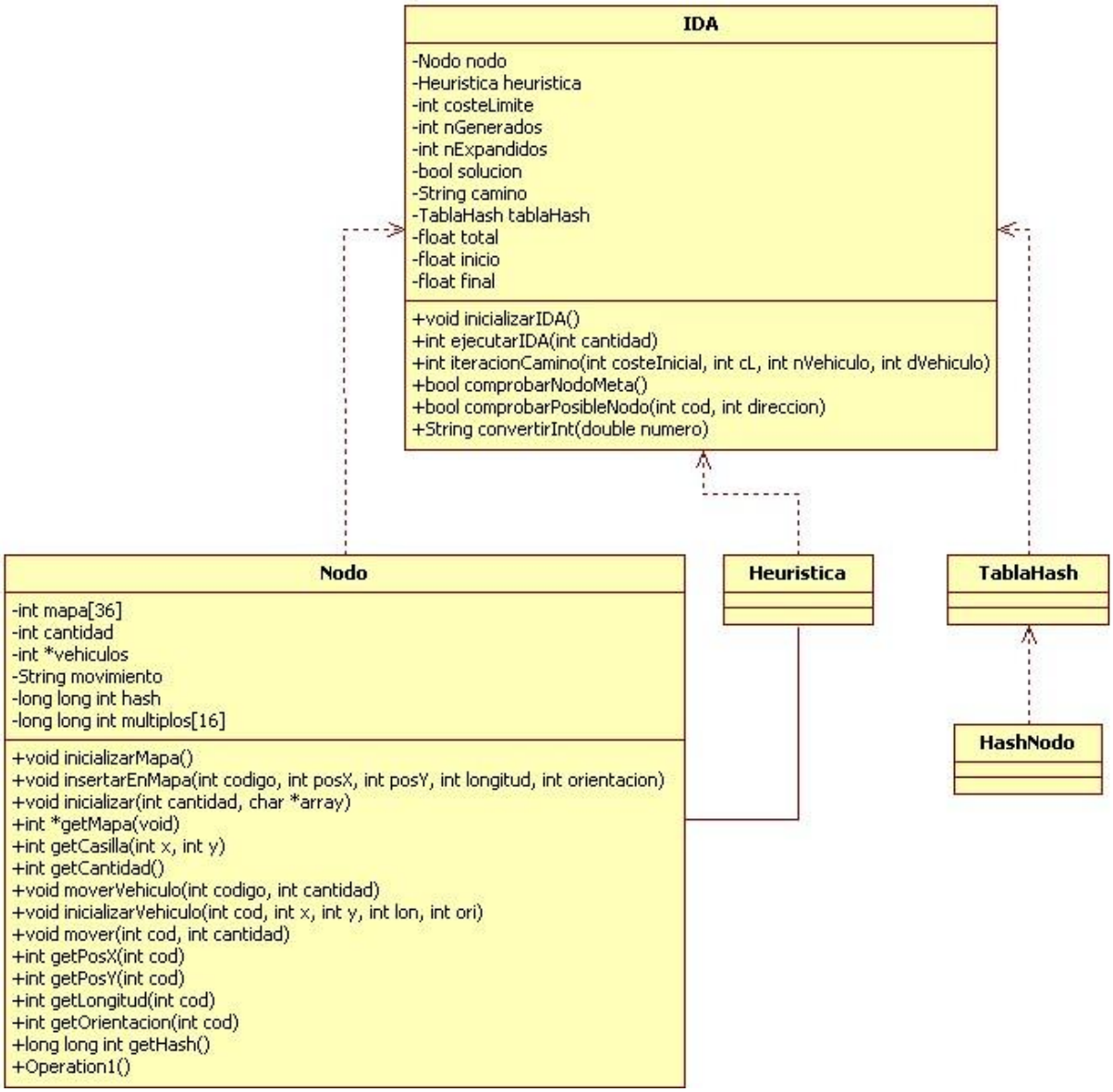


Ilustración 51 - Diagrama completo (clases IDA y Nodo)

Modelos 3D

En este apartado se explica define brevemente las técnicas utilizadas para la realización de los modelos 3D, así como una imagen de cada uno de ellos.

Blender

Para comenzar, se define con brevedad la interfaz y cada uno de los apartados más importantes del software, el cual es totalmente configurable (accesos directos y disposición de la pantalla) para una mayor comodidad del usuario.

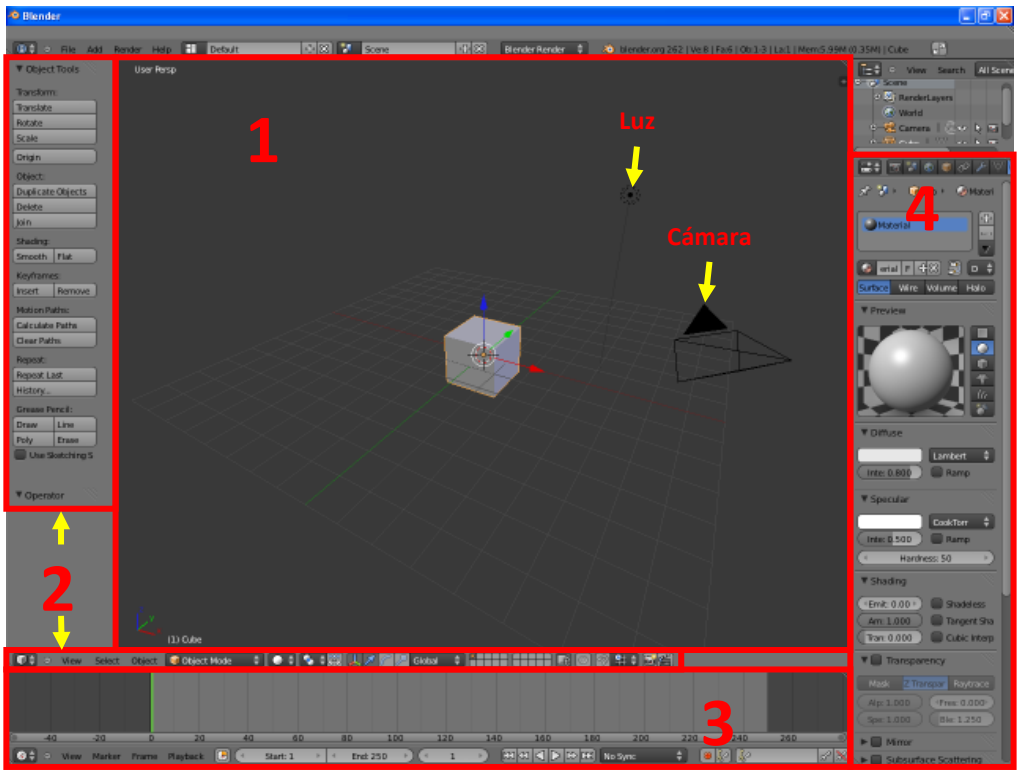


Ilustración 52 - Interfaz Blender

1- **Vista 3D**: Esta parte de la ventana es donde se insertan y modifican los diferentes modelos. Se puede configurar para tener varias vistas desde distinta posición cada una e incluso se puede ver el objeto en perspectiva u ortogonalmente.

2- **Operaciones**: En estas secciones de la ventana se pueden realizar distintas operaciones sobre los objetos o configurar algunos aspectos del uso.

3- **Timeline**: Para las animaciones 3D.

4- **Propiedades**: Son las propiedades de los objetos, uno de los aspectos más importantes de la edición de modelos 3D y el aplicado de texturas y materiales.

Técnicas de modelado

Blender ofrece una gran cantidad de posibilidades a la hora de modelar, ya sea en cuanto a tipo de modelado como a herramientas para manipular los modelos.

Forma de modelar

Para realizar un modelo, se ha partido de una forma poligonal rígida y se ha ido editando polígono a polígono, aplicando espejo para equilibrar la edición a ambos lados (ya que son modelos simétricos con respecto a uno de los ejes).

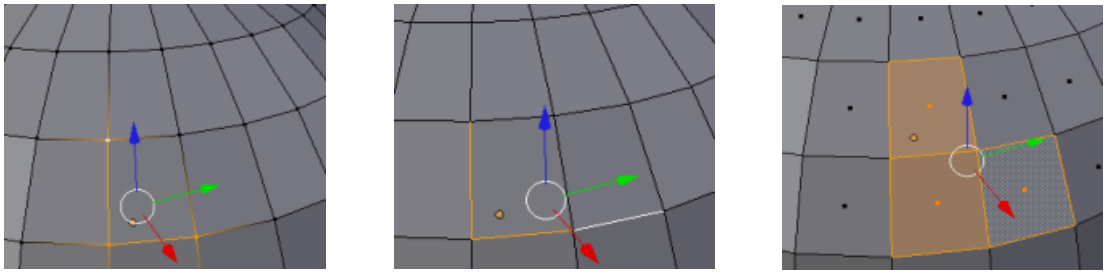


Ilustración 53 - Edición de vértices, aristas y caras

Herramientas

Las herramientas que se han utilizado para los modelos han sido las siguientes:

- **Extrude**: Esta herramienta sirve básicamente para "levantar" una cara (o varias, según hallamos seleccionado).

Esto no es lo mismo que realizar una transformación sobre una cara (o varias), ya que mover dichas caras hacia afuera provocaría que las caras colindantes se inclinaran para conseguir la continuidad. Lo que se realiza, además de mover dichas caras seleccionadas, es la adición de nuevas caras que conecten con las colindantes.

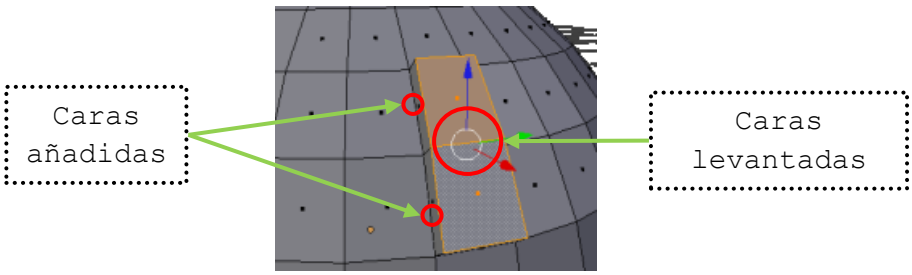


Ilustración 54 - Herramienta *extrude*

- **Merge**: Esta herramienta nos permite unir un vértice B a un vértice A eliminando el vértice B y uniendo todos los "vecinos" (caras y aristas) de B a A.

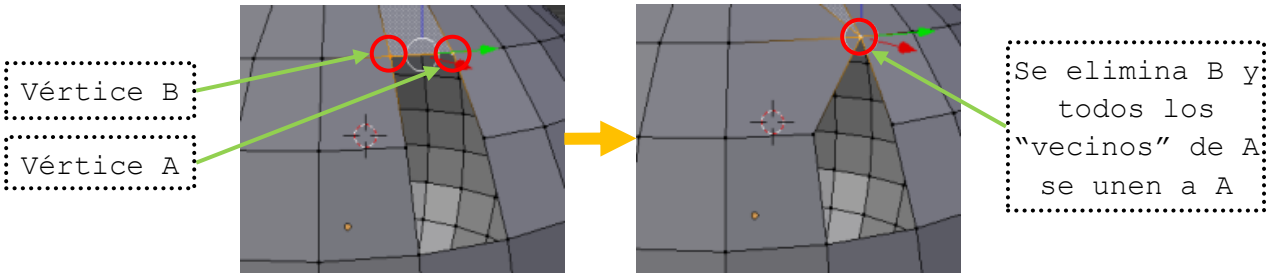


Ilustración 55 - Herramienta *merge*

- **Crease**: Esta herramienta es imprescindible para usar correctamente la *Subdivision Surface* (explicado más adelante). Lo que hace es aumentar la "rigidez" de una arista.

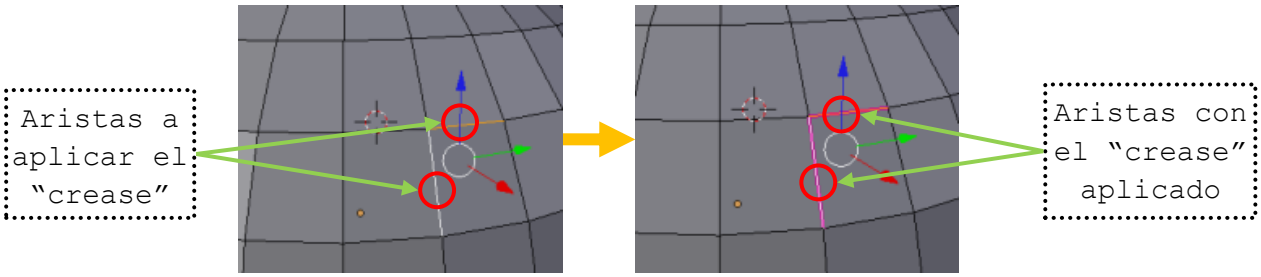


Ilustración 56 - Herramienta *crease*

- **Subdivide**: La última herramienta importante que se ha utilizado ha sido aquella que divide una cara en dos o más. Para ello, hay que seleccionar al menos dos de las aristas que limitan dicha cara.

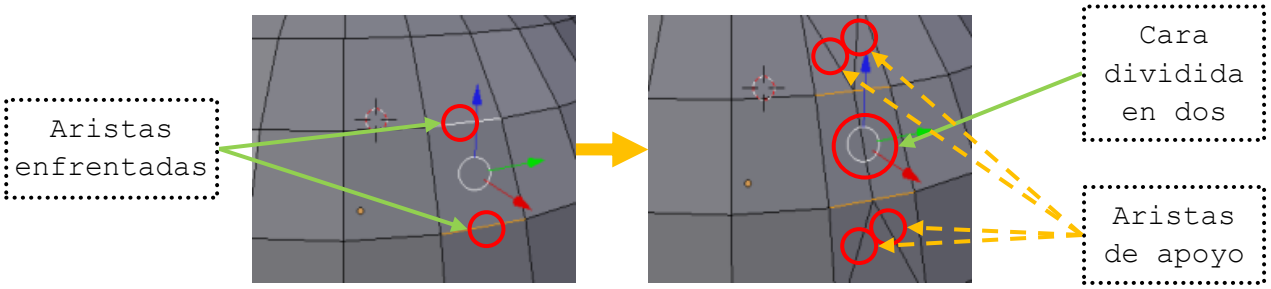


Ilustración 57 - Herramienta *subdivide*

Por último, se va a proceder a explicar el *Subdivision Surface* y la importancia de aplicar correctamente la herramienta crease.

Subdivisión surface

Esta técnica consiste en el suavizado de los objetos mediante la adición de nuevas caras de forma que se cada polígono se subdividan en 4.

Esta técnica requiere una computación elevada ya que el número de caras aumenta exponencialmente dependiendo del número de iteraciones que se quieran realizar. Así pues, si tenemos una sola cara, con 3 iteraciones tendríamos 1 * 4 * 4 * 4 = 64 caras; con 10 iteraciones tendríamos 1.048.576 caras. Si esto se aplica a un objeto que, por ejemplo, tenga 100 caras, tendríamos 6400 caras con 3 iteraciones y 104.857.600 caras con 10.

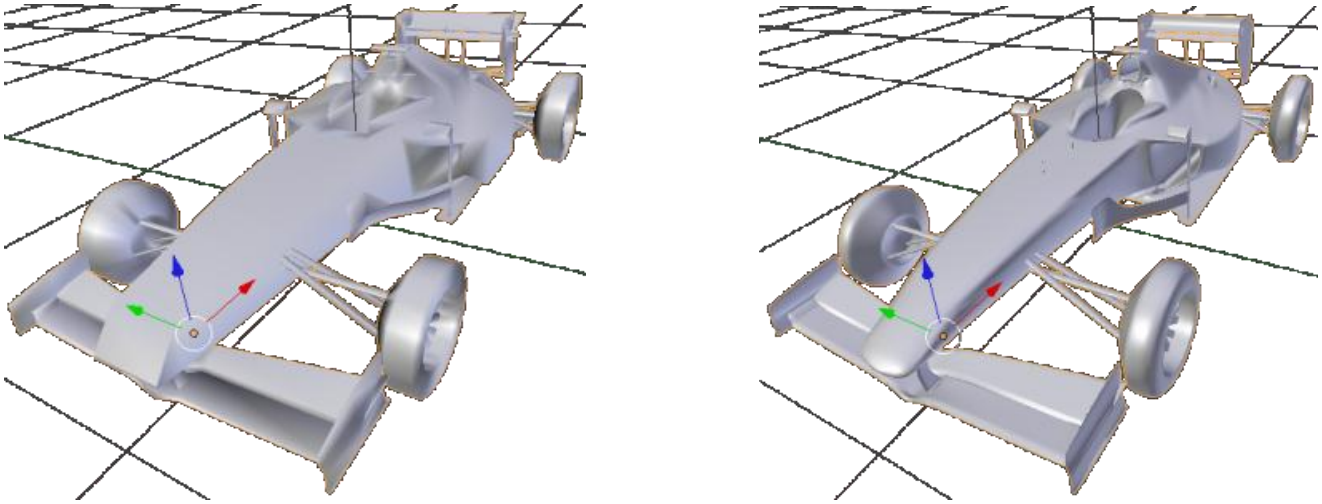


Ilustración 58 - Izq) Sin *subdivision surface*. Der) Con *subdivision surface* de 3

Importancia del crease

Es posible que haya partes del modelo que no se quieran suavizar, o al menos no demasiado. Esto se soluciona con la aplicación de la herramienta crease en determinadas aristas. Esta herramienta tiene un nivel del 0 al 1 de aplicación.

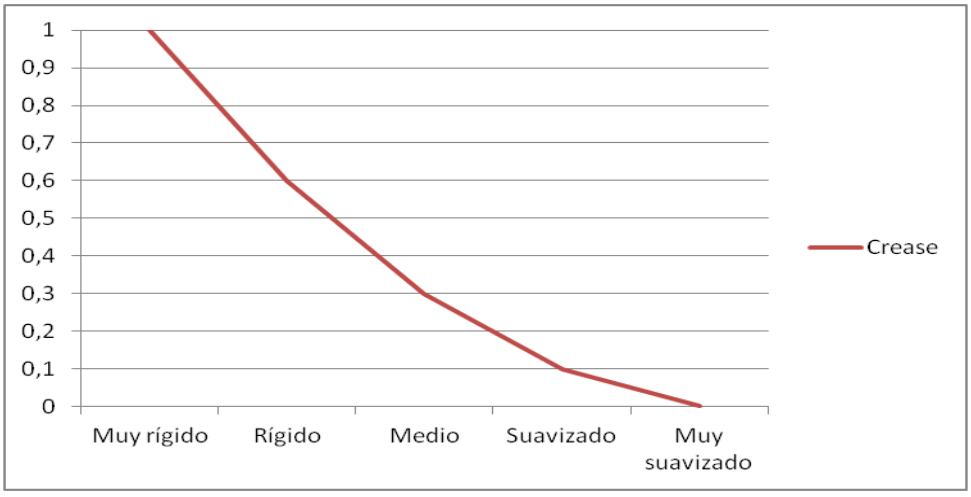


Ilustración 59 - Proporción del *crease* y el suavizado

A continuación se muestra un ejemplo de la diferencia entre usar un crease de 0.75 y no usarlo sobre un cubo suavizado con 5 iteraciones de Subdivision Surface:

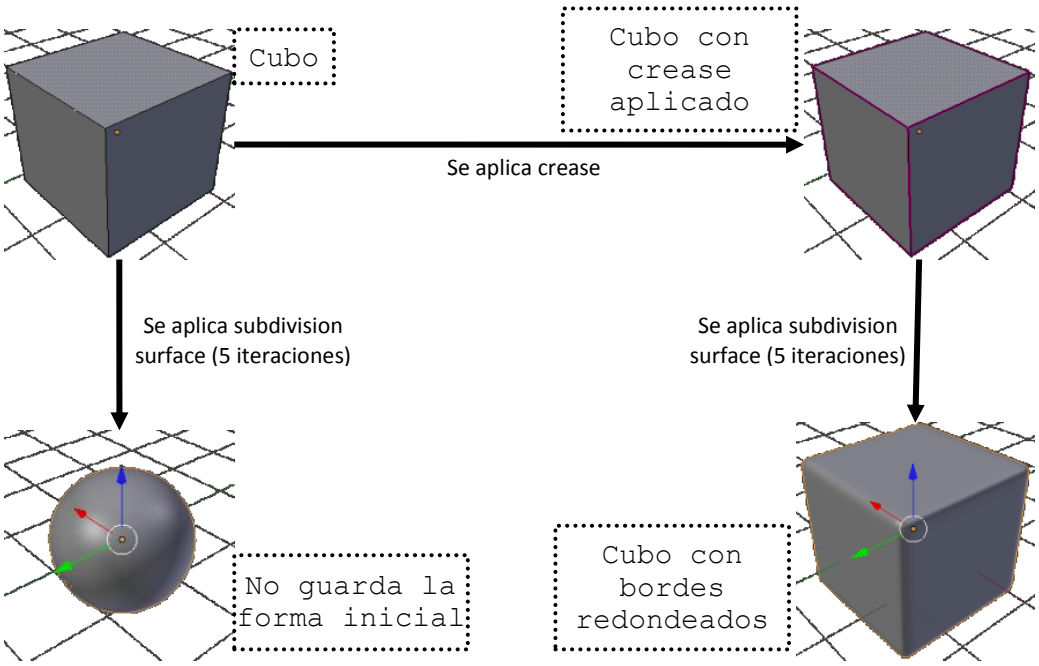


Ilustración 60 - Diferencia entre aplicar o no la herramienta "crease" antes del suavizado

Texturizado

En cuanto al texturizado de los modelos, se ha realizado mediante mapas UV e imágenes creadas con GIMP.

Los mapas de textura UV no son más que una configuración para trasladar partes de la textura a partes del modelo mediante una transformación 2D (UV) a 3D (XYZ). Esta configuración se hace a mano y puede resultar bastante complicada en un principio.

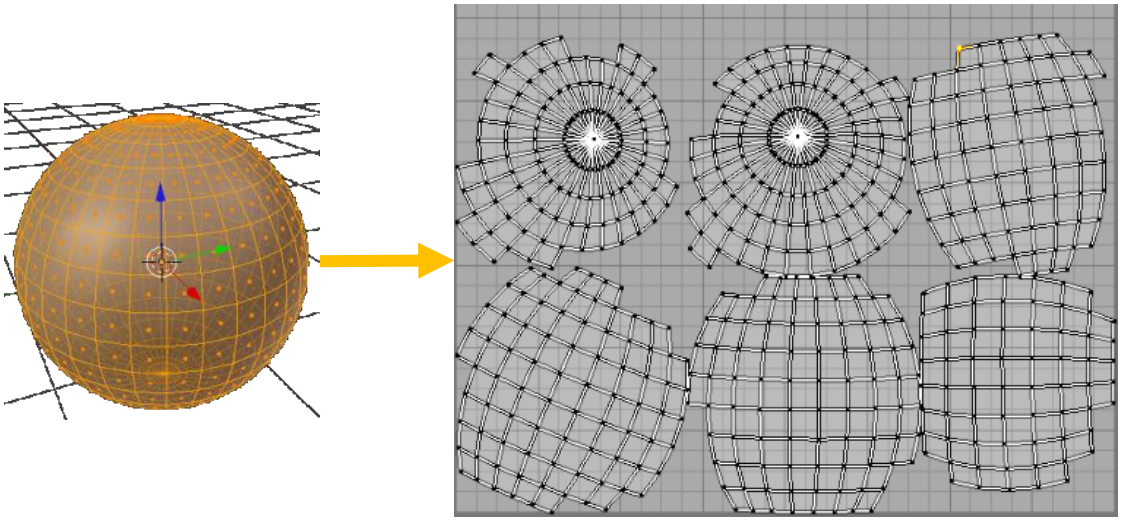


Ilustración 61 - Conversión de un modelo 3D a coordenadas UV

Modelos finales

A continuación, tenemos los tres modelos finales más importantes utilizados en este proyecto.

Vehículo principal

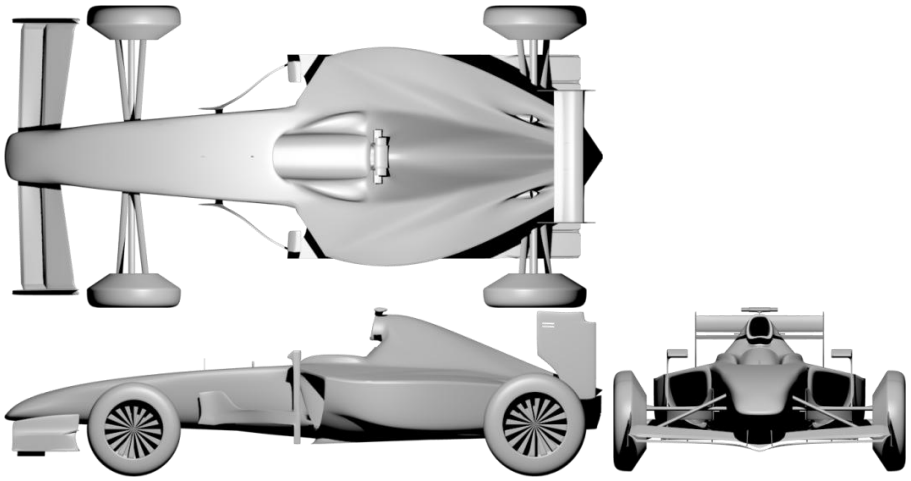


Ilustración 62 - Vistas top, side y front del vehículo principal

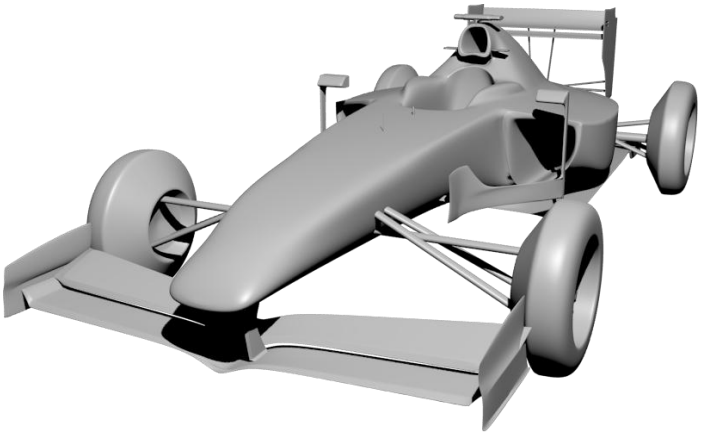


Ilustración 63 - Modelo en perspectiva del vehículo principal

Ilustración 64 - Modelo texturizado del vehículo principal

Coche

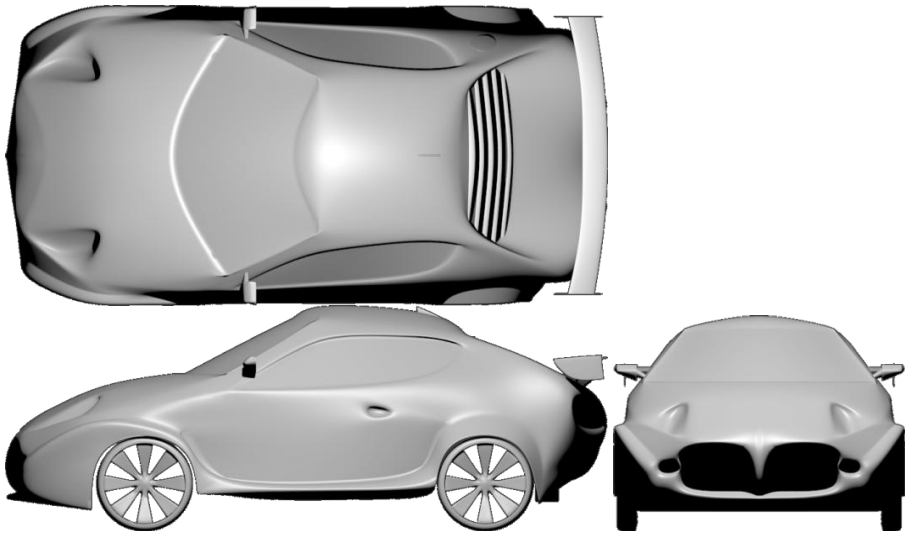


Ilustración 65 - Vistas top, side y front de los coches

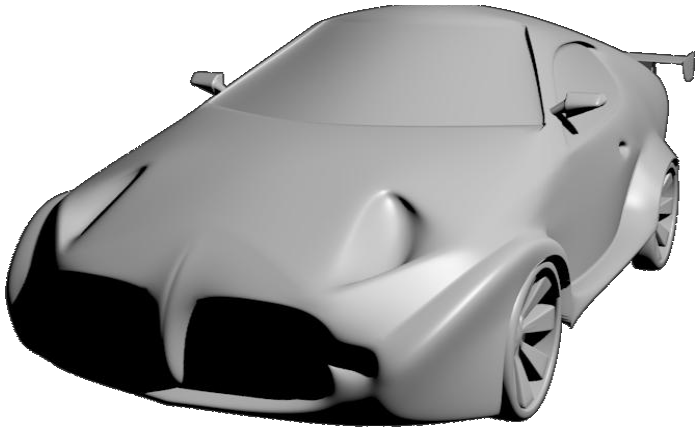


Ilustración 66 - Modelo en perspectiva de los coches

Ilustración 67 - Modelo texturizado de un coche

Camión

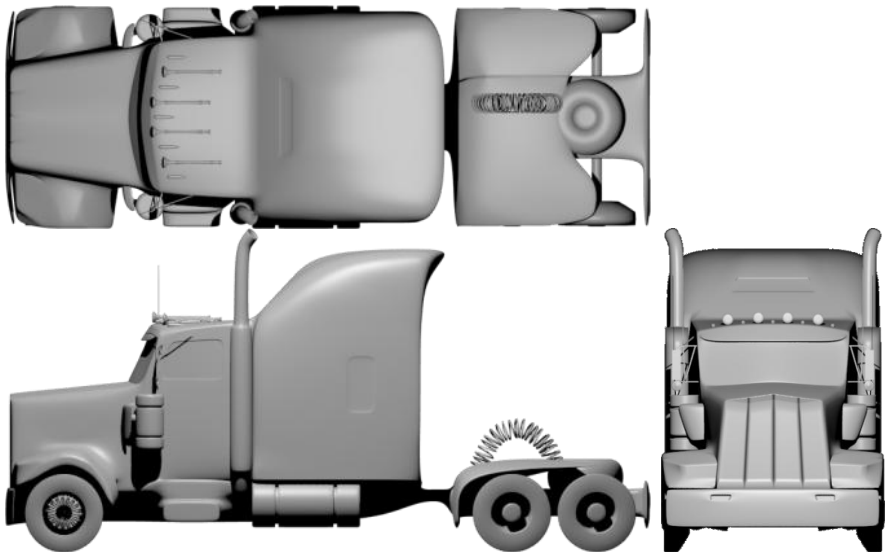


Ilustración 68 - Vistas top, side y front de los camiones

Ilustración 69 - Modelo en perspectiva de los camiones

Ilustración 70 - Modelo texturizado de un camión

Capítulo 5

Pruebas y resultados

En esta sección del documento se procede a explicar cada una de las pruebas realizadas y a exponer unos resultados junto a las conclusiones obtenidas de ellos.

Estas pruebas han sido realizadas en un PC con un procesador AMD FX-8120 (8 núcleos a 3.1/4.0 GHz), 8GB de RAM DDR3 a 1600MHz y una tarjeta gráfica ATI RADEON HD 6870, con Windows 7 Professional .

Las pruebas están separadas en 3 secciones:

- **Número de nodos generados**: Es el número de nodos totales que se han generado a partir de nodos anteriores (incluyendo el nodo raíz). En resumen es la suma de los nodos expandidos más los nodos hoja.

- **Número de nodos expandidos**: Es el número de nodos de los cuales se ha obtenido nodos hijo, es decir, los nodos no hoja.

- **Tiempo**: Es el mejor tiempo obtenido hasta el momento. El tiempo es variable ya que depende del tipo de procesador, velocidad y ociosidad durante la ejecución del programa. Por lo tanto, no hay un tiempo óptimo, simplemente se actualiza el resultado cuando se obtiene un tiempo menor al anterior.

Se han realizado estas pruebas sobre los 40 tableros de la baraja del Rush Hour original separadas en los cuatro niveles de dificultad. Además, se ha realizado las pruebas sobre dos heurística sencillas para comprobar las diferencias entre ambas. Estas heurísticas proporcionan de menos a más información sobre el problema respectivamente.

- **Heurística 1**: Es la distancia del coche principal hasta el nodo meta.

$$h1(n) = 4 - x_{principal}$$

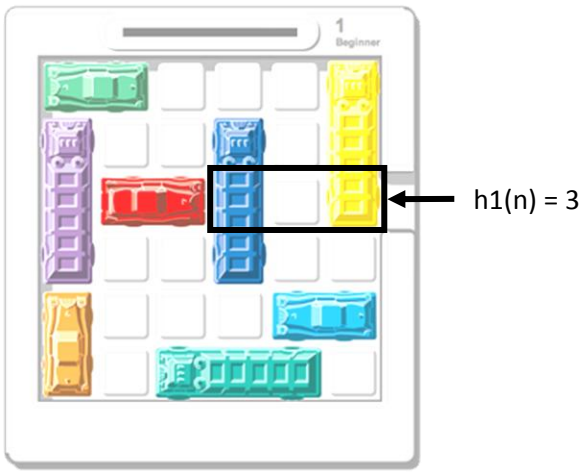


Ilustración 71 – Ejemplo de la heurística 1

- **Heurística 2**: Es la distancia del coche principal hasta el nodo meta más el número de vehículos que hay en su camino.

$$h2(n) = h1(n) + \sum_{posX=x_{principal+2}}^{5} \begin{cases} 1, & tablero(posX, 3) > 0 \\ 0, & si\ no \end{cases}$$

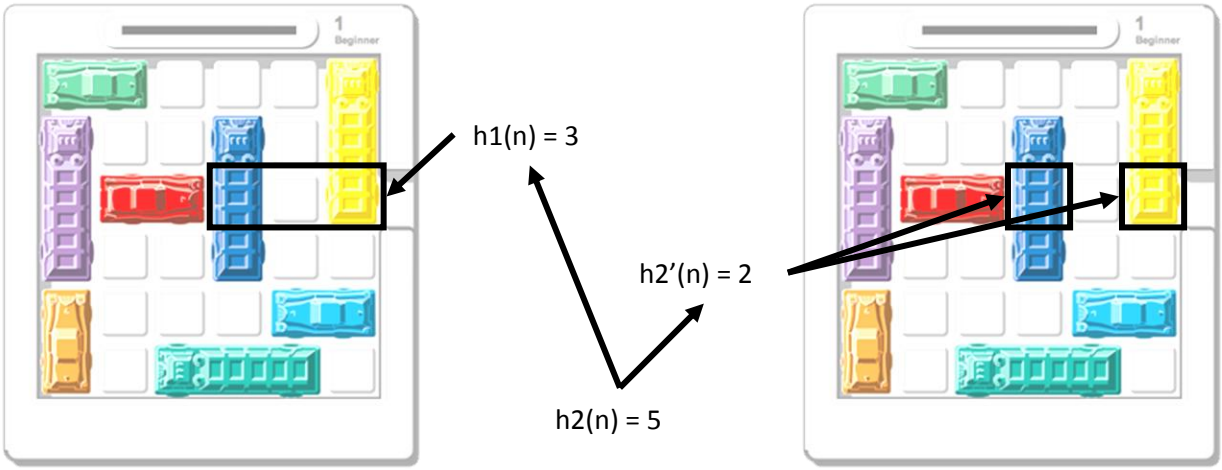


Ilustración 72 - Ejemplo de la heurística 2

Nodos generados

A continuación se muestran las cuatro tablas de los resultados para los nodos generados:

Nodos generados	01	02	03	04	05	06	07	08	09	10
Heurística 1	43715	47081	44809	9553	97615	82103	114619	41381	6001	123463
Heurística 2	31479	18146	37942	6625	55575	48596	89132	27366	2788	111073

Tabla 26 - Nodos generados para los tableros 01-10

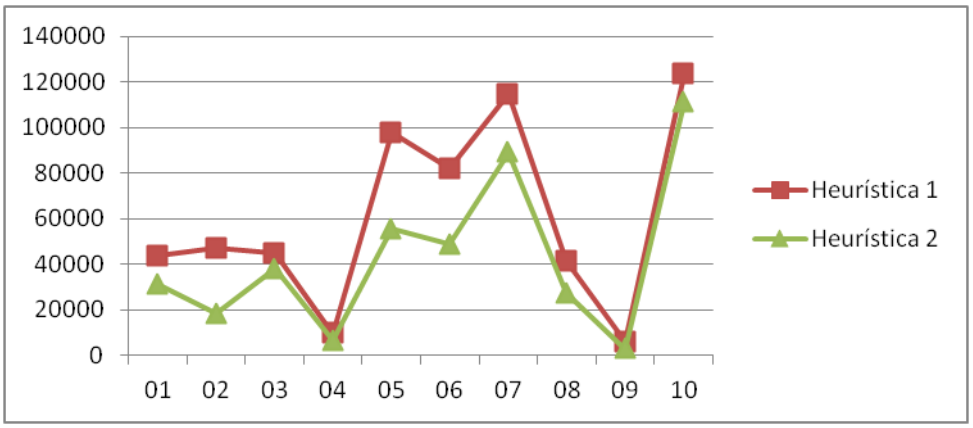


Ilustración 73 - Gráfica de nodos generados para los tableros 01-10

Nodos generados	11	12	13	14	15	16	17	18	19	20
Heurística 1	89634	31701	652764	520314	45471	243222	232000	186469	55157	21617
Heurística 2	83134	30451	481006	391443	40776	239712	205195	175896	55393	11537

Tabla 27 - Nodos generados para los tableros 11-20

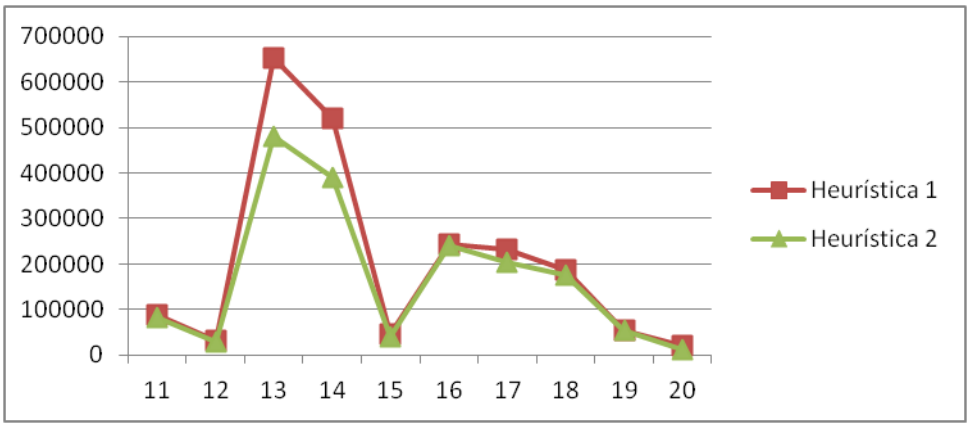


Ilustración 74 - Gráfica de nodos generados para los tableros 11-20

Nodos generados	21	22	23	24	25	26	27	28	29	30
Heurística 1	21081	225277	100554	853389	1012650	549165	193709	99720	648751	101893
Heurística 2	20161	187119	75175	787314	832376	493458	153248	73480	579037	93935

Tabla 28 - Nodos generados para los tableros 21-30

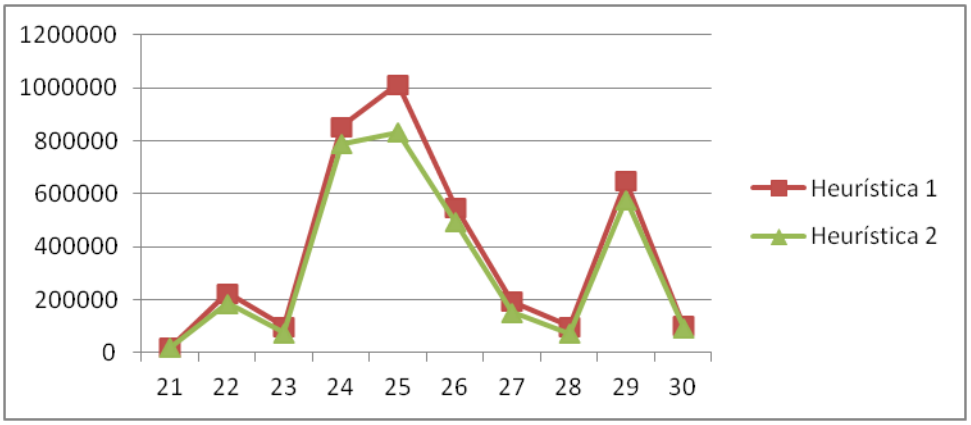


Ilustración 75 - Gráfica de nodos generados para los tableros 21-30

Nodos generados	31	32	33	34	35	36	37	38	39	40
Heurística 1	493022	65527	446017	644454	648458	267584	168927	346629	688157	329489
Heurística 2	456258	64603	410452	590114	624375	262810	128837	310965	645894	305612

Tabla 29 - Nodos generados para los tableros 31-40

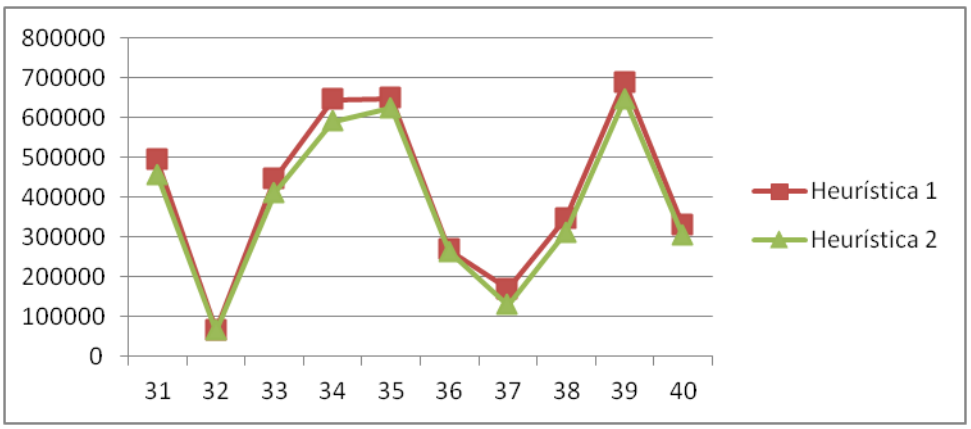


Ilustración 76 - Gráfica de nodos generados para los tableros 31-40

Nodos expandidos

A continuación se muestran las cuatro tablas de los resultados para los nodos expandidos:

Nodos expandidos	01	02	03	04	05	06	07	08	09	10
Heurística 1	6184	5999	7785	1831	12973	11178	15394	6941	1016	19298
Heurística 2	4459	2340	6615	1293	7379	6517	12024	4601	510	17378

Tabla 30 - Nodos expandidos para los tableros 01-10

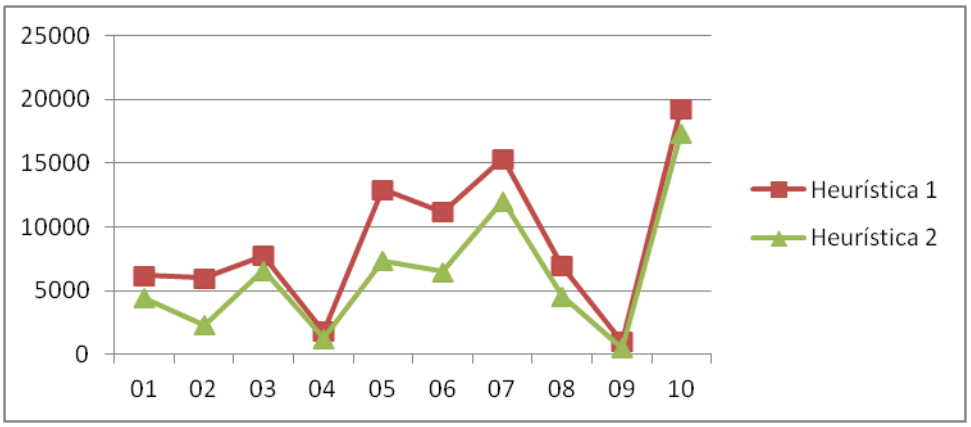


Ilustración 77 - Gráfica de nodos expandidos para los tableros 01-10

Nodos expandidos	11	12	13	14	15	16	17	18	19	20
Heurística 1	17030	5798	89860	69126	8231	36551	35004	33096	9897	3792
Heurística 2	15801	5455	67197	53019	7382	35658	30953	31210	9757	2084

Tabla 31 - Nodos expandidos para los tableros 11-20

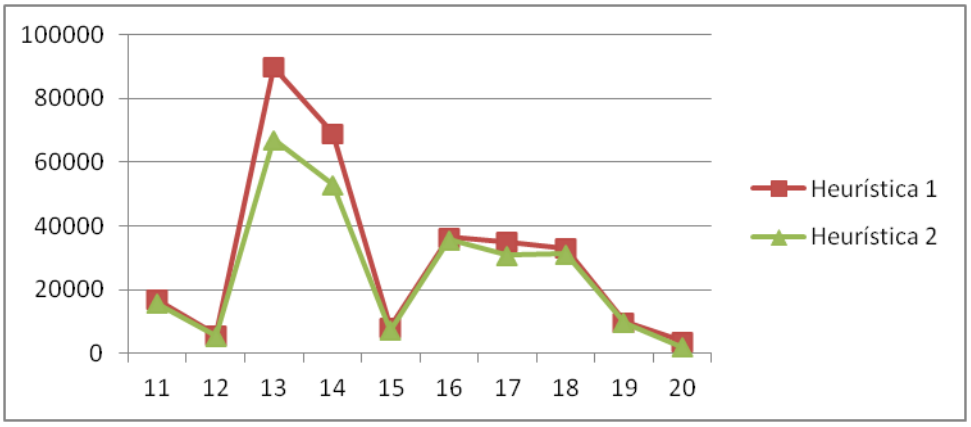


Ilustración 78 - Gráfica de nodos expandidos para los tableros 11-20

Nodos expandidos	21	22	23	24	25	26	27	28	29	30
Heurística 1	5093	34449	19146	112479	132414	78179	35567	17029	92639	19135
Heurística 2	4867	29368	14670	103697	109504	70090	28596	12980	82657	17680

Tabla 32 - Nodos expandidos para los tableros 21-30

Ilustración 79 - Gráfica de nodos expandidos para los tableros 21-30

Nodos expandidos	31	32	33	34	35	36	37	38	39	40
Heurística 1	79712	14312	72387	102714	111059	43705	28229	62174	122913	55789
Heurística 2	73908	14071	66338	94794	106793	42912	22173	56061	115361	52135

Tabla 33 - Nodos expandidos para los tableros 31-40

Ilustración 80 - Gráfica de nodos expandidos para los tableros 31-40

Tiempo

A continuación se muestran las cuatro tablas de los resultados para los mejores tiempos:

Tiempo (s)	01	02	03	04	05	06	07	08	09	10
Heurística 1	0,027	0,048	0,023	0,01	0,09	0,082	0,107	0,052	0,012	0,118
Heurística 2	0,022	0,029	0,023	0,009	0,06	0,058	0,085	0,04	0,008	0,105

Tabla 34 - Tiempo de cómputo para los tableros 01-10

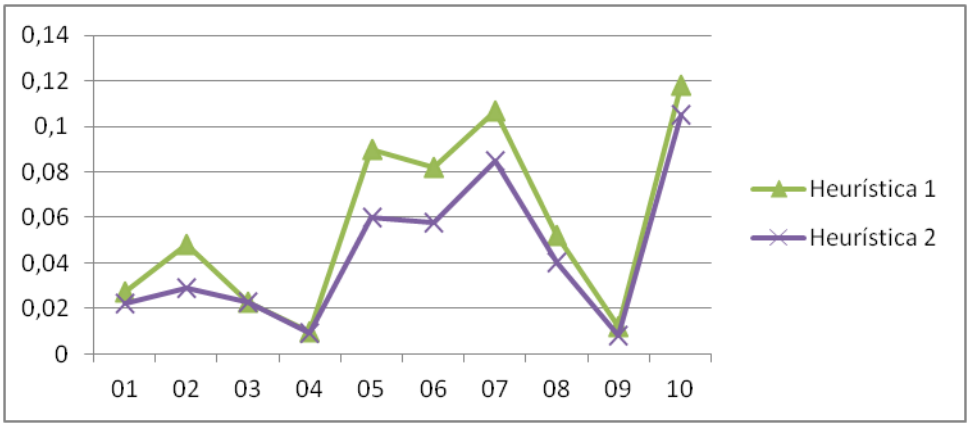


Ilustración 81 - Gráfica de tiempo de cómputo para los tableros 01-10

Tiempo	11	12	13	14	15	16	17	18	19	20
Heurística 1	0,039	0,022	0,609	0,473	0,047	0,221	0,189	0,16	0,025	0,028
Heurística 2	0,034	0,023	0,454	0,35	0,042	0,202	0,176	0,14	0,038	0,024

Tabla 35 - Tiempo de cómputo para los tableros 11-20

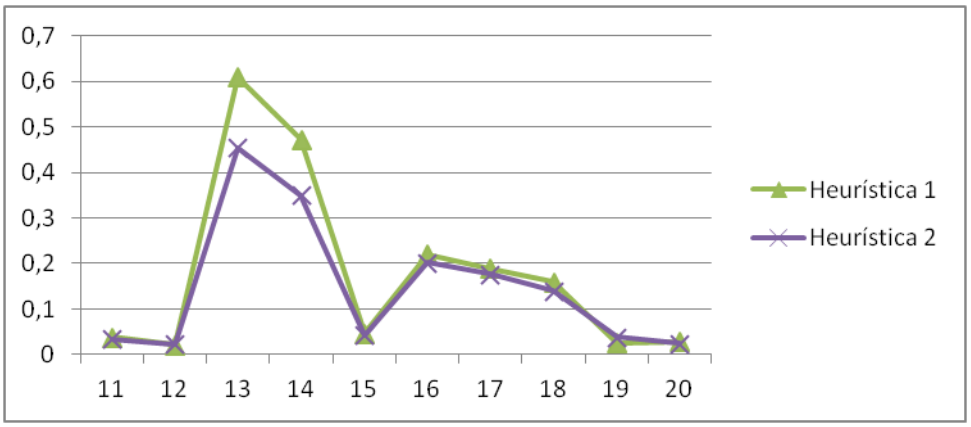


Ilustración 82 - Gráfica de tiempo de cómputo para los tableros 11-20

Tiempo	21	22	23	24	25	26	27	28	29	30
Heurística 1	0,021	0,187	0,093	0,697	0,91	0,472	0,176	0,089	0,553	0,094
Heurística 2	0,016	0,164	0,079	0,648	0,748	0,418	0,145	0,074	0,49	0,091

Tabla 36 - Tiempo de cómputo para los tableros 21-30

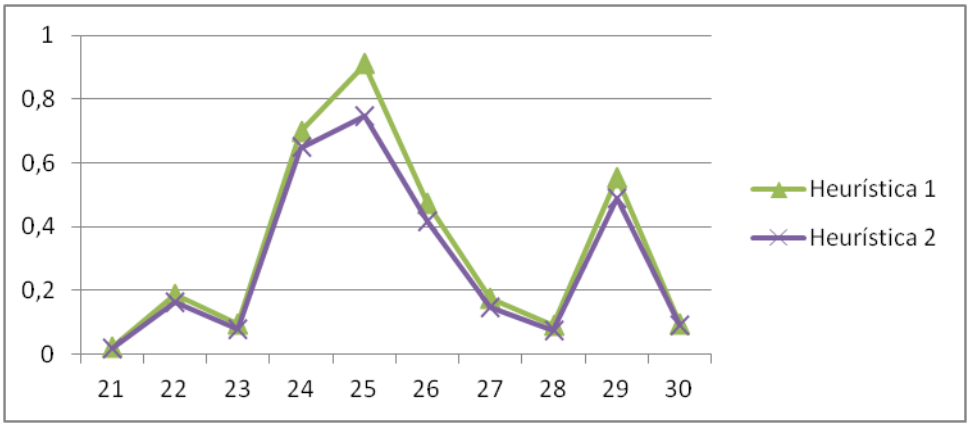


Ilustración 83 - Gráfica de tiempo de cómputo para los tableros 21-30

Tiempo	31	32	33	34	35	36	37	38	39	40
Heurística 1	0,436	0,067	0,559	0,559	0,561	0,217	0,15	0,305	0,606	0,278
Heurística 2	0,387	0,06	0,358	0,556	0,562	0,22	0,117	0,285	0,542	0,256

Tabla 37 - Tiempo de cómputo para los tableros 31-40

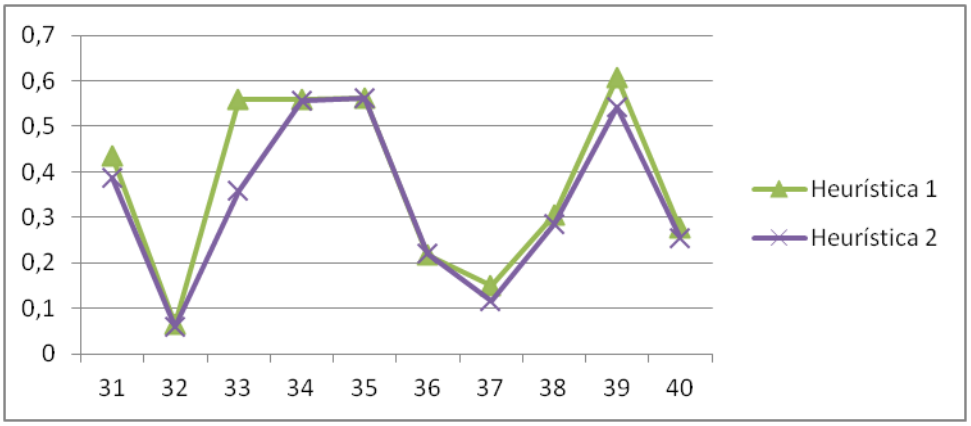


Ilustración 84 - Gráfica de tiempo de cómputo para los tableros 31-40

Conclusiones sobre los resultados

Como podemos observar, los resultados muestran que el tiempo consumido incluso en los tableros más complicados son muy buenos, no superando el segundo de tiempo de cómputo en ninguno de los casos.

Un algoritmo de inteligencia artificial como el IDA*, además de darnos un buen tiempo de ejecución (aunque no sea el más rápido), nos da la posibilidad de realizar problemas muy grandes sin aumentar el coste espacial nada más que para la tabla de transposición. Por este motivo, el IDA* es uno de los algoritmos más utilizados en la actualidad, sobre todo para problemas que con otros algoritmos no se podrían resolver.

En cuanto a las heurísticas se puede observar que la segunda es mejor que la primera al ser una heurística más informada. Concretamente la mejora en nodos generados, nodos expandidos y tiempo de ejecución es la siguiente:

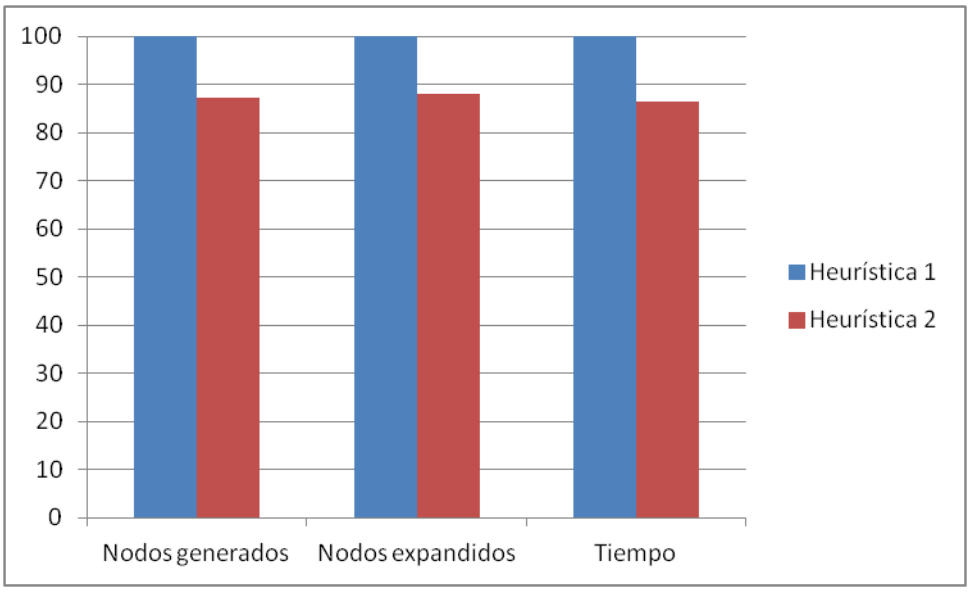


Ilustración 85 - Coste de la heurística 2 con respecto a la heurística 1

Capítulo 6

Conclusiones

En este capítulo se muestra un repaso de los objetivos para comprobar la consecución de los mismos, así como una enumeración de los problemas encontrados a la hora de realizar el proyecto.

Objetivos alcanzados

Los objetivos seguidos para la realización de este proyecto se encuentran en el capítulo 3, concretamente en la página XX. A continuación se enumeran estos objetivos y se exponen las conclusiones pertinentes para cada uno de ellos.

- **Subsistemas**: El sistema se dividirá en dos partes importantes: La interfaz estará programada en Python mientras que el/los algoritmo(s) de búsqueda y heurística(s) estarán programados en C++ para conseguir una mayor velocidad en la ejecución.

La división del sistema en dos grandes subsistemas, programados en Python y C++ para la interfaz y el agente respectivamente, y la posibilidad de comunicarse entre ambos, se ha realizado satisfactoriamente gracias a las librerías SWIG que permiten la comunicación entre módulos programados en distintos lenguajes.

Además, la alta velocidad de C++ hace que se resuelvan rápidamente las instancias de Rush Hour de 6x6.

- **Solución de instancias de 6x6**: El sistema será capaz de resolver instancias del juego Rush Hour de tamaño 6x6 con tamaño de vehículos estándar (2x1 y 3x1).

El sistema es capaz de resolver cualquier instancia de Rush Hour para tableros de 6x6 y de forma óptima. Para comprobar esto se han creado los 40 tableros del Rush Hour original y otros tableros inventados para poner a prueba al sistema.

También se han realizado tableros sin solución para comprobar que el sistema informa de ello.

- **Selección de instancias por características**: El usuario indicará al sistema qué instancia quiere que se resuelva. Dichas instancias estarán ordenadas por dificultad.

En el menú principal, al seleccionar el submenú de “Resolver” o de “Editor” se puede seleccionar, además, una de las cuatro dificultades (Fácil, Normal, Difícill y Experto). Los diferentes tableros están ordenados por nivel de dificultad.

Además, en el caso del editor, un tablero se guarda en el nivel de dificultad seleccionado.

- **Creación de nuevas instancias**: El usuario podrá crear una nueva instancia, mediante un editor, la cual podrá seleccionar posteriormente para que el sistema la resuelva. El sistema no realizará una comprobación de si la instancia es resoluble o no.

Con el editor se puede crear tableros. Seleccionando el tablero "Nuevo" de cualquier nivel de dificultad se abrirá el editor con el tablero vacío y todos los coches disponibles.

- **Edición de instancias**: El usuario podrá editar una instancia ya creada, mediante una interfaz gráfica, la cual podrá seleccionar posteriormente para que el sistema la resuelva. El sistema no realizará una comprobación de si la instancia es resoluble o no.

Con el editor se puede editar tableros. Seleccionando cualquier tablero previamente guardado de cualquier nivel de dificultad se abrirá el editor con el tablero ocupado por los coches del tablero a editar y sólo estando disponibles los vehículos no utilizados en el tablero.

- **Salvado de instancias**: El usuario podrá guardar en un fichero la nueva instancia creada o editada para su posterior resolución.

Efectivamente, se puede guardar los tableros en un fichero con extensión .rh. La sobreescritura de ficheros no está permitida, esto es debido al deseo de evitar que se borre alguno de los 40 tableros originales.

- **Representación gráfica**: El sistema representará los movimientos que pertenecen a la solución de la instancia de forma gráfica.

No sólo se ha representado gráficamente los movimientos de la solución de un tablero, sino que, además, la interfaz entera está representada de forma gráfica en 3D con animaciones (como el movimiento de las ruedas de los vehículos, etc.).

- **Obtención de estadísticas**: El sistema mostrará las estadísticas de cada instancia (número de nodos generados, número de nodos expandidos, tiempo empleado, etc).

Durante la selección del tablero (tanto para resolver como para editar) se muestran las mejores estadísticas recogidas hasta el momento para cada uno de ellos.

También se muestran las estadísticas al finalizar la animación que muestra cómo se resuelve el tablero. Estas estadísticas se muestran en rojo si son peores que las mejores hasta el momento, en naranja si son iguales o en verde si son mejores.

Problemas encontrados

Básicamente, los problemas encontrados durante la realización de este proyecto han sido los del coste de aprendizaje del software de apoyo utilizado (el motor gráfico Panda3D y SWIG) ya que nunca antes había utilizado estas librerías ni ninguna parecida.

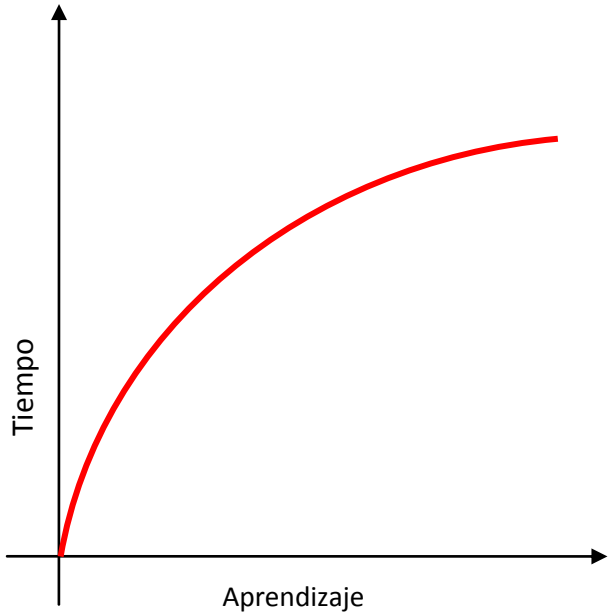


Ilustración 86 - Curva de aprendizaje de las librerías

Además, la implementación de las tablas de transposición surgió a partir de serios problemas que consistían en que la mayor parte de los tableros tardaba muchísimo en resolverse, del orden de decenas de minutos. Una vez realizada la tabla de transposición, todos los tableros del Rush Hour original se resolvían en menos de 2 segundos (dependiendo de las características del ordenador donde se ejecutase el programa).

Finalmente, otro de los problemas con respecto a la interfaz es el descenso esporádico de los cuadros por segundo (Frames Per Second) para tableros con una gran cantidad de vehículos (8 o más) debido a la ralentización del sistema por la cantidad de polígonos que la tarjeta tiene que mover. Esto también depende de la tarjeta gráfica ya que tras realizar pruebas en distintos ordenadores se ha comprobado que, los descensos de FPS, no ocurre en las tarjetas gráficas potentes.

Conclusiones finales

Tras realizar un gran esfuerzo en todo lo comentado en este documento se ha conseguido un proyecto que funciona satisfactoriamente cumpliendo todos y cada uno de los requisitos comentados en el capítulo 3.

El agente realizado ha servido como base para un estudio sobre el comportamiento del algoritmo de búsqueda IDA* sobre los problemas NP-Completo. La posibilidad de ampliar el problema de Rush Hour a tableros mayores (10x10, 15x15, etc) con nuevos tipos de vehículos, e incluso la posibilidad de realizar un agente que genere tableros resolubles de forma aleatoria, amplia el área de estudio en lo que se refiere a este juego.

Se han adquirido además nuevos conocimientos tanto de lenguajes de programación (nunca había programado en Python) como de uso de librerías (motor gráfico y librería de conexión entre diferentes lenguajes).

Capítulo 7

Líneas futuras

En esta sección se describe una serie de mejoras o líneas futuras por las que se puede continuar el proyecto.

Mejora del rendimiento gráfico

Una buena mejora sería modificar los modelos 3D de los vehículos para reducir significativamente el número de polígonos utilizados en ellos sin reducir la calidad visual. Esto haría que aumentara el número de cuadros por segundo en cualquier tipo de PC.

El problema viene dado por las técnicas de alisado de los modelos *subdivision surface* que cuatriplica el número de polígonos por cada iteración ésta:

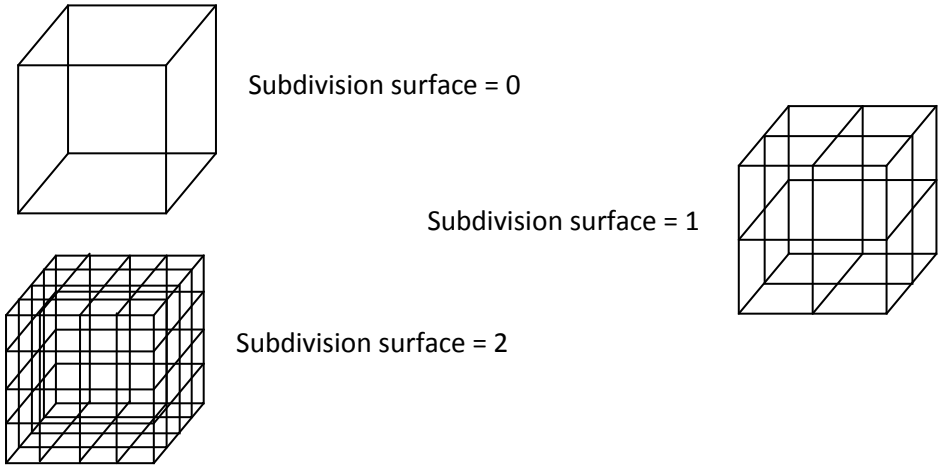


Ilustración 87 – Creación de nuevas caras con el subdivision surface

En la siguiente gráfica podemos ver el número de caras que obtendríamos del primer cubo dependiendo de la iteración de *subdivision surface*. Es claramente visible la relación exponencial.

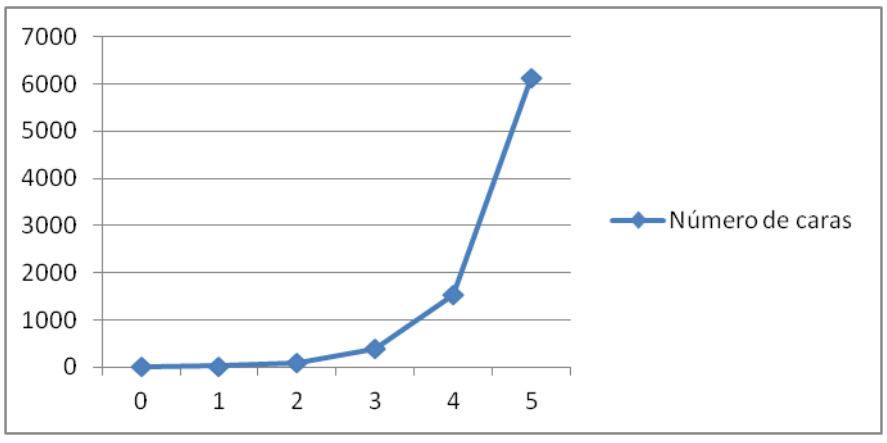


Ilustración 88 - Gráfica del crecimiento de caras con respecto al número de iteraciones de subdivision surface

Para evitar esto se realizará un modelado más preciso intentando alisar lo máximo posible el modelo antes de aplicar la técnica de *subdivision surface* para evitar entre 1 y 2 iteraciones.

Generador automático de instancias

Una forma de poder estudiar casos fuera de las 40 instancias que vienen con el juego original sería la realización de un sistema que, con el mismo algoritmo de búsqueda IDA*, pudiera generar instancias distintas del juego.

Aumento del tamaño del tablero

Otra posibilidad sería la de aumentar el tamaño del tablero, hasta 10x10 o incluso más.

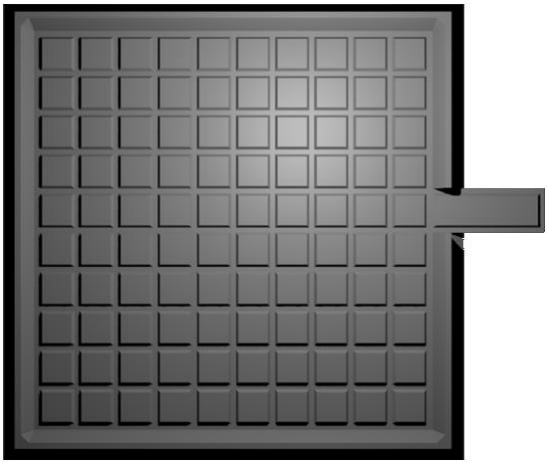


Ilustración 89 - Tablero de 10x10

La complejidad del problema aumentaría considerablemente aunque se seguiría pudiendo resolver debido a la utilización del algoritmo IDA* que evita el crecimiento de los recursos de espacio utilizados independientemente del tamaño del problema.

Para esto, sería necesario modificar el formato de las claves hash ya que actualmente utiliza variables de tipo "long long int", debido a que ofrece hasta números de 128 bits. Sin embargo, con un tablero de 10x10 y suponiendo una proporción de vehículos similar (75% de coches y 25% de camiones, aproximadamente serían 32 coches y 12 camiones) tendríamos una cantidad de posibles configuraciones de tablero muy grandes, aproximadamente de:

$$9'7 * 10^{41}$$

Lo que implicaría disponer de un número de 140 bits. La solución es realizar una función hash con una cadena de caracteres como clave. Se seguiría utilizando el árbol de transposiciones para guardar la información relativa al valor heurístico de cada nodo ya explorado.

Nuevos tipos de vehículo

Un aumento del tamaño del tablero implicaría la posibilidad de introducir nuevos tipos de vehículo de mayor longitud. En dicho caso, no sería necesario realizar modificaciones en el agente ya que el algoritmo de búsqueda está implementado para recoger los "choques" entre vehículos de forma genérica, es decir, independientemente de su longitud. Sin embargo habría que realizar grandes modificaciones en la interfaz para el editor.

Adaptación de nuevos módulos

Éste quizás sea una de las líneas más importantes para el futuro de este proyecto en el que se reestructuraría el código de tal manera que sería posible la inclusión de nuevos algoritmos de búsqueda y heurísticas.

Básicamente, los posibles algoritmos de búsqueda y heurísticas a utilizar (actualmente sólo está implementado el IDA* con 3 heurísticas) se "capturarían" dinámicamente ya que estarían guardados en distintos ficheros con unos parámetros de entrada/salida iguales para todos ellos.

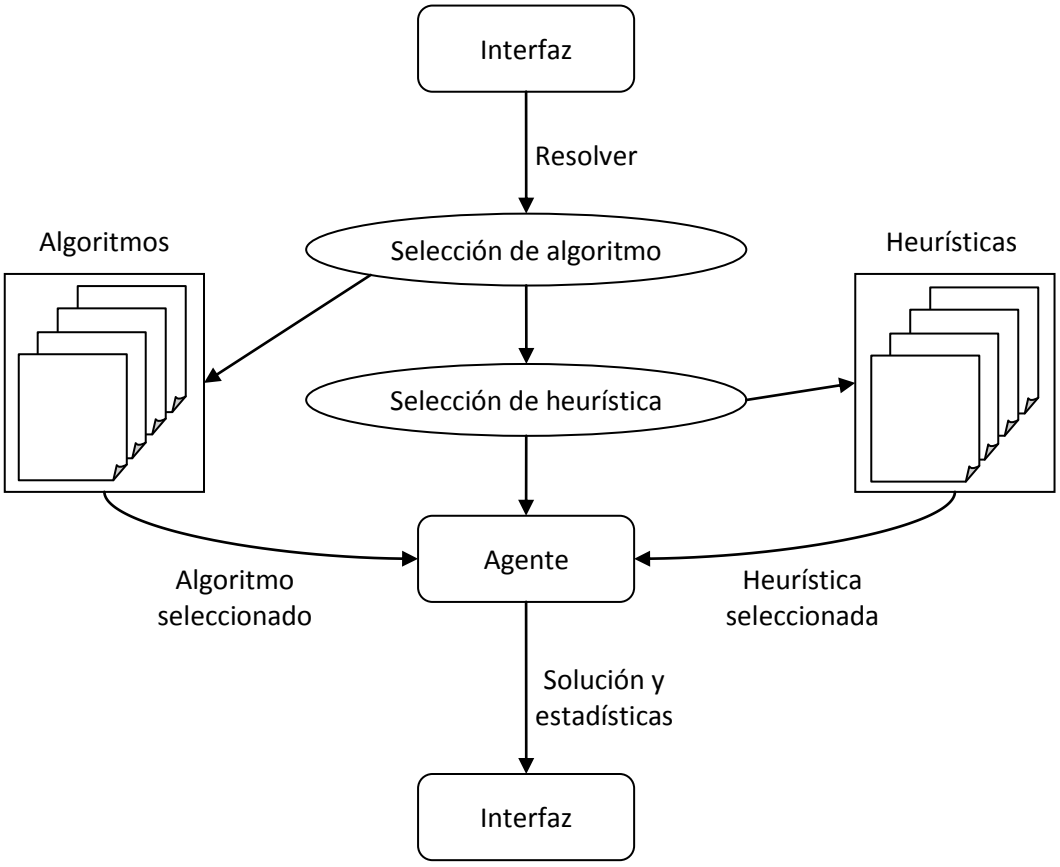


Ilustración 90 - Adquisición dinámica de algoritmos y heurísticas

Anexo A

Planificación y presupuesto

En este primer anexo se muestra la planificación realizada previamente a la realización del proyecto junto a los costes asociados a personal y herramientas (tanto hardware como software).

Planificación

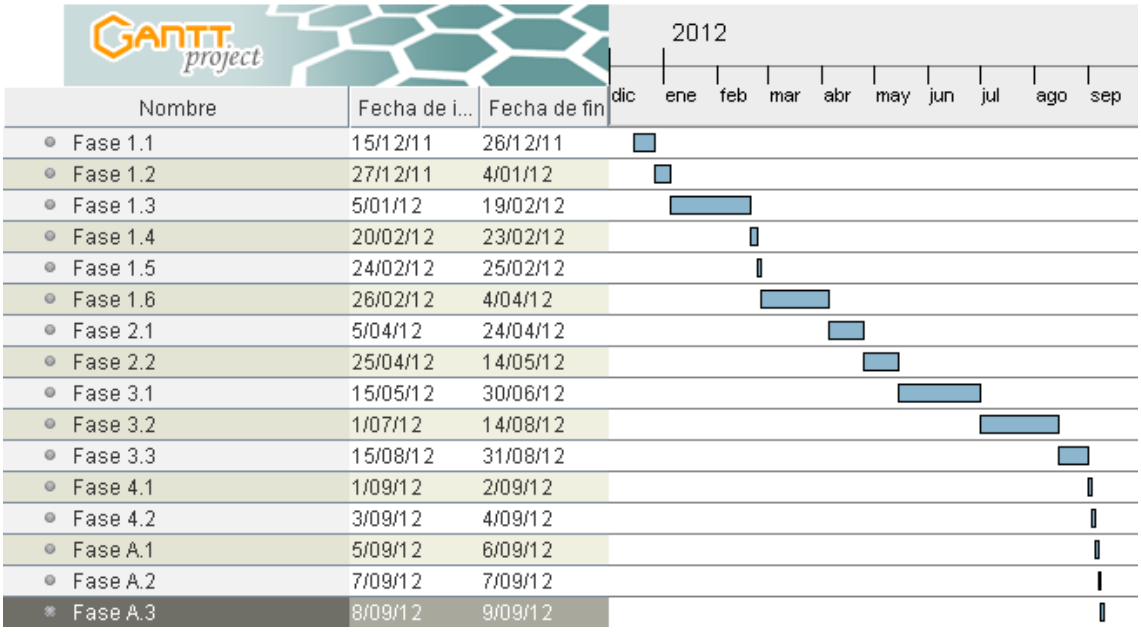
Basicamente, todos los proyectos se pueden dividir en cuatro grandes apartados: Analisis, diseño, programación y pruebas. Además, estos apartados se pueden dividir en otros subapartados más específicos.

A continuación se muestra un listado de las distintas etapas del proyecto, considerando un ciclo de vida en cascada:

Planificación	
Etapa 1	Análisis.
Fase 1.1	Estudio de los distintos motores gráficos.
Fase 1.2	Estudio de los distintos elementos de IA (algoritmos de búsqueda, heurísticas y tablas de transposición).
Fase 1.3	Extracción de requisitos.
Fase 1.4	Toma de decisiones en cuanto a elementos, tanto de la IA como del apartado gráfico.
Fase 1.5	Planificación y presupuesto estimado.
Fase 1.6	Escritura del documento.
Etapa 2	Diseño.
Fase 2.1	Diseño arquitectónico (diagrama de clases) de la aplicación.
Fase 2.2	Realización de los modelos 3D (modelado y texturizado).
Etapa 3	Programación.
Fase 3.1	Programación del interfaz.
Fase 3.2	Programación del agente.
Fase 3.3	Conexión entre interfaz y agente.
Etapa 4	Pruebas.
Fase 4.1	Planificación de las pruebas.
Fase 4.2	Realización de las pruebas.
Anexo	Aprobación del proyecto
Fase A.1	Introducción de los resultados de las pruebas y conclusiones.
Fase A.2	Revisión ortográfica del documento.
Fase A.3	Revisión de contenidos del documento.

Tabla 38 - Planificación de tareas

La siguiente gráfica de Gantt recoge las fechas de inicio, fin y la duración de cada una de las fases del proyecto (incluyendo fines de semana y festivos):



Nombre	Fecha de i...	Fecha de fin
Fase 1.1	15/12/11	26/12/11
Fase 1.2	27/12/11	4/01/12
Fase 1.3	5/01/12	19/02/12
Fase 1.4	20/02/12	23/02/12
Fase 1.5	24/02/12	25/02/12
Fase 1.6	26/02/12	4/04/12
Fase 2.1	5/04/12	24/04/12
Fase 2.2	25/04/12	14/05/12
Fase 3.1	15/05/12	30/06/12
Fase 3.2	1/07/12	14/08/12
Fase 3.3	15/08/12	31/08/12
Fase 4.1	1/09/12	2/09/12
Fase 4.2	3/09/12	4/09/12
Fase A.1	5/09/12	6/09/12
Fase A.2	7/09/12	7/09/12
Fase A.3	8/09/12	9/09/12

Ilustración 91 - Gráfica de Gantt

En total se ha estimado una duración de 270 días que, calculado a 4 horas por día, resulta de 1080 horas.

Presupuesto

En cuanto a los costes presupuestados, en la siguiente tabla se recogen ordenados por categorías según el tipo de recurso.

Presupuesto				
Recurso	**Coste**	**Vida útil estimada**	**Uso estimado**	**Coste final**
Hardware				
PC para el desarrollo	870€	72 meses	270 días	108,75€
Software				
Windows 7 Professional	250€	120 meses	270 días	18,75€
Blender	0€	-	-	0€
Gimp	0€	-	-	0€
Panda3D	0€	-	-	0€
Gedit	0€	-	-	0€
Dev-C++	0€	-	-	0€
Personal				
Analista, diseñador, programador	15.000€ (4 horas/día)	12 meses	270 días	11250€
Otros recursos				

Material de oficina	40€	-	-	40€
Total				11.417,50€

Tabla 39 - Presupuesto inicial

Finalmente, los costes reales se han visto ligeramente aumentados ya que el proyecto se ha finalizado el día 03 de octubre. Por lo tanto, el presupuesto real es el siguiente:

		Presupuesto		
Recurso	**Coste**	**Vida útil estimada**	**Uso estimado**	**Coste final**
Hardware				
PC para el desarrollo	870€	72 meses	294 días	118,41€
Software				
Windows 7 Professional	250€	120 meses	294 días	20,41€
Blender	0€	-	-	0€
Gimp	0€	-	-	0€
Panda3D	0€	-	-	0€
Gedit	0€	-	-	0€
Dev-C++	0€	-	-	0€
Personal				
Analista, diseñador, programador	15.000€ (4 horas/día)	12 meses	294 días	12.082,19€
Otros recursos				
Material de oficina	40€	-	-	40€
Total				12.261,01€

Tabla 40 - Presupuesto real

Anexo B

Pseudocódigos

En este anexo se muestran los pseudocódigos de los 5 algoritmos de búsqueda mostrados en el capítulo 2.

Primero en amplitud

```
- Crear lista ABIERTA con el nodo inicial, I, (estado-
  inicial)
- EXITO=Falso
- Hasta que ABIERTA esté vacía O ÉXITO
    o Quitar de ABIERTA el primer nodo, N
    o Si N tiene sucesores
    o Entonces Generar los sucesores de N
        ▪ Crear punteros desde los sucesores hacia N
        ▪ Si algún sucesor es nodo meta Entonces
          EXITO=Verdadero
        ▪ Si no Añadir los sucesores al final de ABIERTA
    o Si ÉXITO Entonces Solución=camino desde I a N por
      los punteros
- Si no, Solución=fracaso
```

Primero en profundidad

Primero en profundidad prácticamente igual que en amplitud, simplemente se introducen los hijos al principio de la cola en vez de al final.

Además, normalmente se indica un nivel máximo de profundidad en el que, a partir de él, no se puede expandir más nodos.

```
- Crear lista ABIERTA con el nodo inicial, I, y su
  profundidad=0
- EXITO=Falso
- Hasta que ABIERTA esté vacía O ÉXITO
    o Quitar de ABIERTA el primer nodo
    o Lo llamaremos N y a su profundidad P
    o Si P < Profundidad-máxima Y N tiene sucesores
      Entonces Generar los sucesores de N
        ▪ Crear punteros desde los sucesores hacia N
        ▪ Si algún sucesor es el Estado-Final
        ▪ Entonces EXITO=Verdadero
        ▪ Si no, Añadir los sucesores al principio de
          ABIERTA y Asignarles profundidad P+1
- Si ÉXITO Entonces Solución=camino desde I a N por los
  punteros
- Si no, Solución=fracaso
```

Primero el mejor

- Crear grafo de búsqueda G, con el nodo inicial, I (Estado-inicial)
- ABIERTA=I, CERRADA=Vacío, EXITO=Falso
- Hasta que ABIERTA esté vacía O ÉXITO
    - Quitar el primer nodo de ABIERTA, N y meterlo en CERRADA
    - SI N es Estado-final ENTONCES EXITO=Verdadero
    - SI NO Expandir N, generando el conjunto S de sucesores de N, que no son antecesores de N en el grafo
        - Generar un nodo en G por cada s de S
        - Establecer un puntero a N desde aquellos s de S que no estuvieran ya en G
        - Añadirlos a ABIERTA
        - Para cada s de S que estuviera ya en ABIERTA o CERRADA: decidir si redirigir o no sus punteros hacia N
        - Para cada s de S que estuviera ya en CERRADA: decidir si redirigir o no los punteros de los nodos en sus subárboles
        - Reordenar ABIERTA según f (n)
- Si EXITO Entonces Solución=camino desde I a N a través de los punteros de G
- Si no Solución=Fracaso

A*

Este algoritmo es igual que el algoritmo de primero el mejor, con excepción de que añade cierta información sobre el coste realizado hasta llegar al nodo. Esto evita el problema de obtener resultados no-óptimos con el algoritmo de primero el mejor.

IDA*

```
   -  EXITO=Falso
   -
   -  Mientras que EXITO=Falso
         o  EXITO=Profundidad (Estado-inicial,)
         o
   -  Solución=camino desde nodo del Estado-inicial al
      Estado-final por los punteros

Profundidad (Estado-inicial,µ)

   -  Expande todos los nodos cuyo coste f(n) no excede µ
```

Anexo C

Manual de usuario

A continuación se muestra un manual de usuario paso a paso para facilitar la familiarización con la herramienta.

Pantalla inicial

Esta es la pantalla inicial que el usuario encuentra al ejecutar el programa.

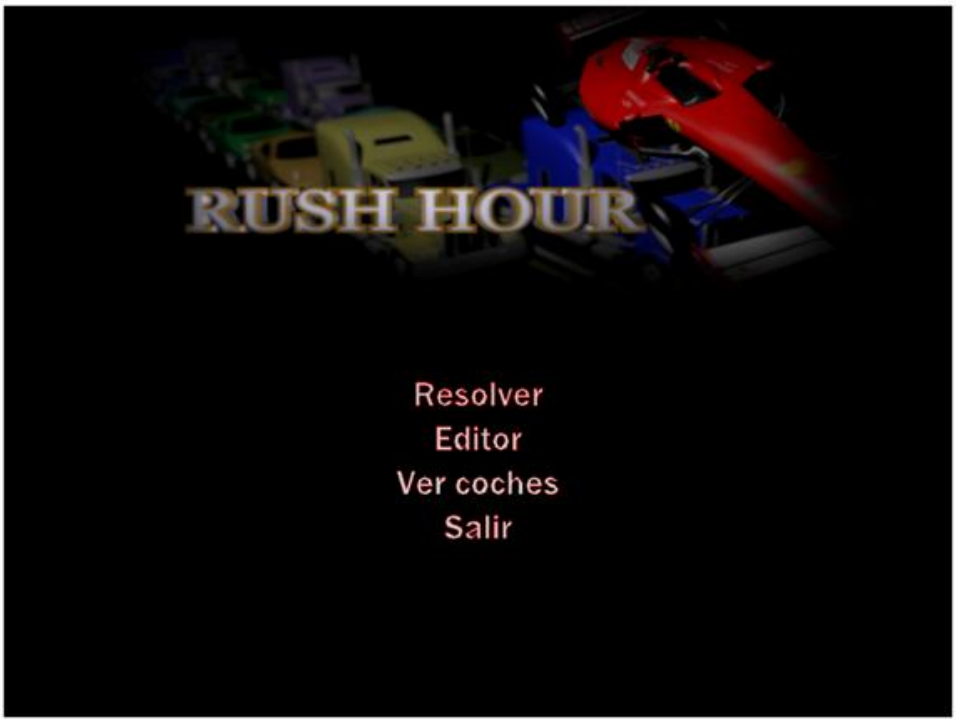


Ilustración 92 - Pantalla de inicio

Navegación por el menú

La navegación por el menú es muy intuitiva y con varias partes muy distinguidas. El usuario se puede encontrar en el menú principal o en una de las funciones que ofrece el software (Resolver, Editor y Ver coches).

Para navegar por el menú principal simplemente hay que hacer click en cualquiera de las opciones mostradas.

Menú

A continuación se muestra cada parte del menú y las acciones que se pueden realizar.

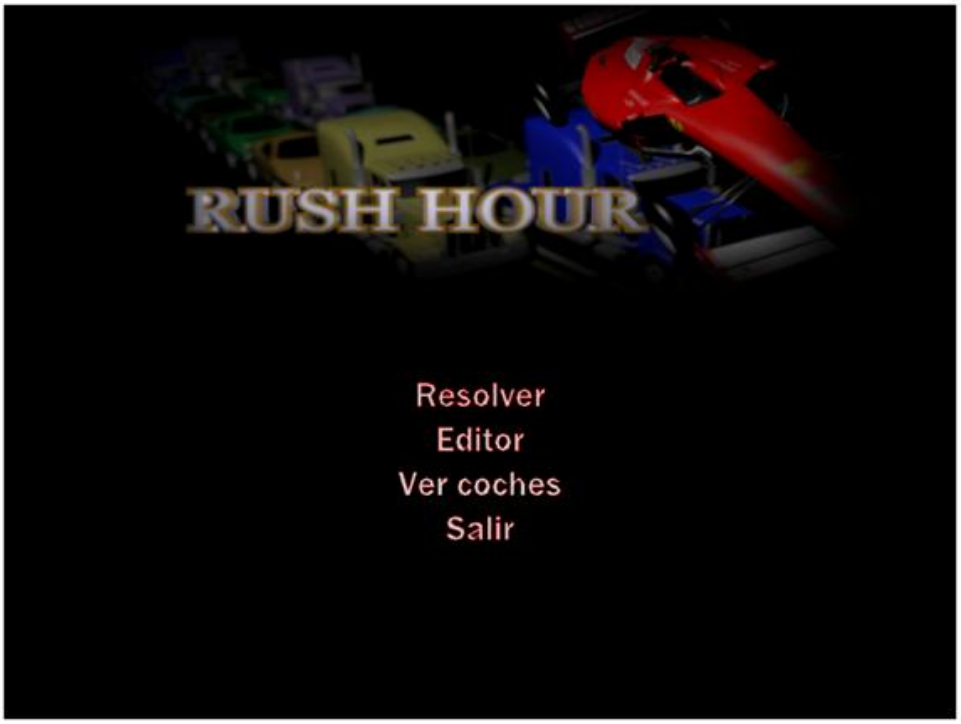


Ilustración 93 - Menú principal

- **Resolver**: Avanza al menú de selección de dificultad para el modo de resolver tablero.
- **Editor**: Avanza al menú de selección de dificultad para el modo de editar/crear tablero.
- **Ver coches**: Permite visualizar los vehículos.
- **Salir**: Finaliza la aplicación.

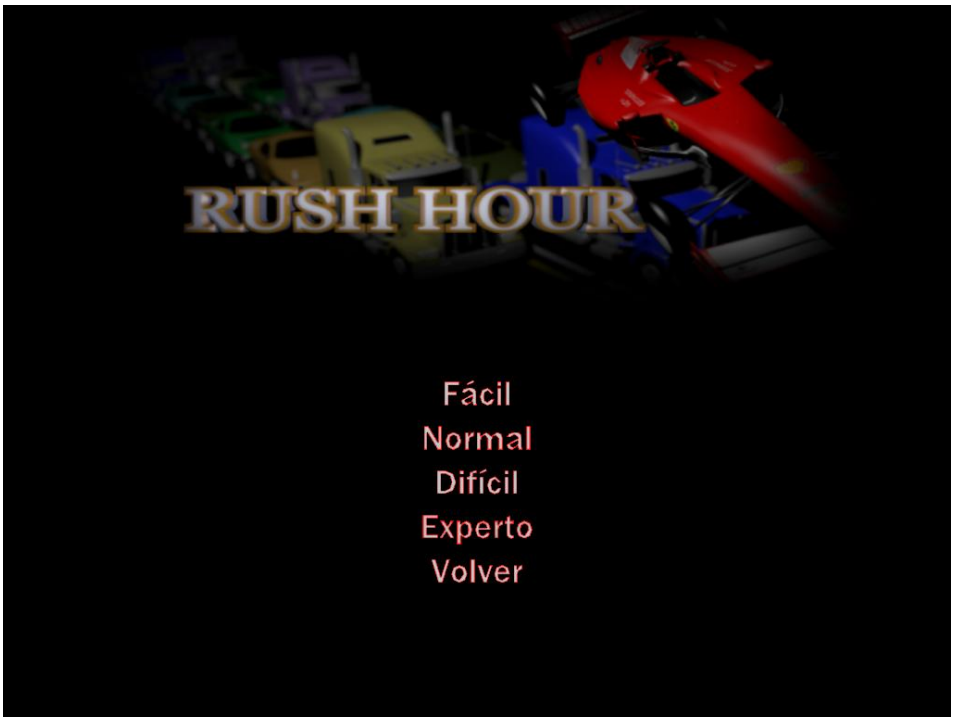


Ilustración 94 - Menú de dificultad

- **Cualquier dificultad**: Selecciona el nivel de dificultad para abrir los tableros a resolver/editar de dicho nivel de dificultad.

- **Volver**: Vuelve al menú inicial.

Ilustración 95 - Menú de selección de tablero

- **Flecha izquierda/derecha**: Cambia de tarjeta.
- **Aceptar**: Acepta el tablero y se abre para editarlo/resolverlo (depende de en qué submenú se encuentre el usuario).
- **Volver**: Vuelve al selector de dificultad.

En este menú se pueden dar dos casos particulares:

- Para crear un tablero desde cero en el editor, existe una tarjeta llamada "Nuevo" sin ningún vehículo en ella.
- En cuanto al otro caso, si en uno de los niveles de dificultad no hay ningún tablero y nos encontramos en el submenú de resolver, aparece un aspa roja indicándoselo al usuario.

Resolver

En el momento en el que se hace click en el botón de aceptar y nos encontramos en el submenú resolver, se calcula los movimientos necesarios y se procede a mostrar la animación de cómo se resuelve. Finalmente aparecen las estadísticas de la ejecución.

Ilustración 96 - Animación de la resolución del tablero

Mientras se muestra la animación, se puede girar el tablero manteniendo el botón izquierdo del ratón presionado y moviéndolo. Una vez soltado el botón volverá a su posición original.

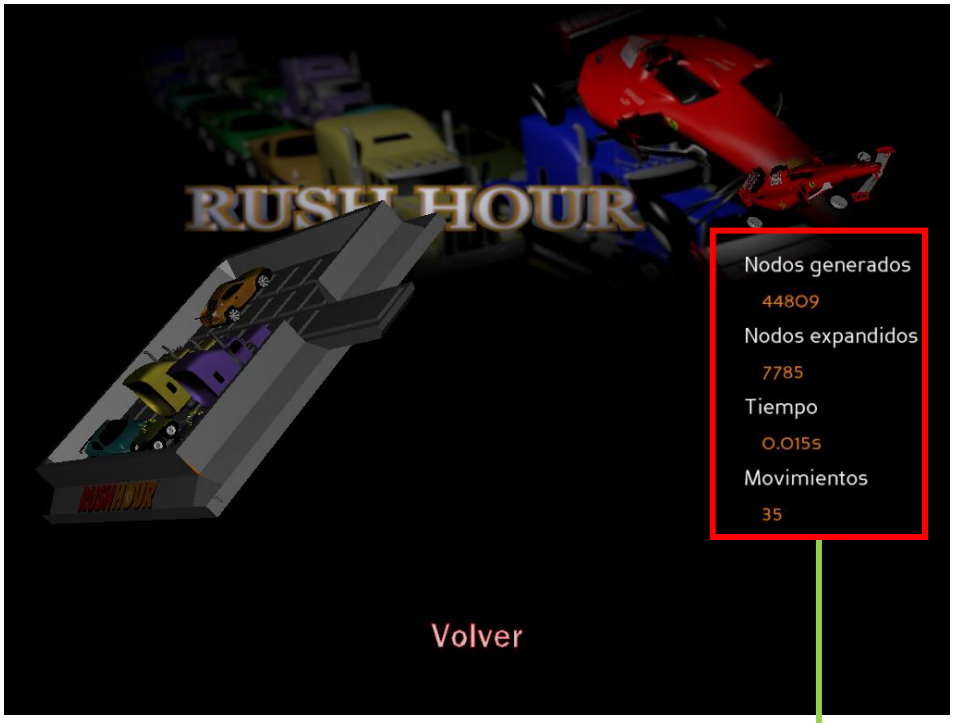


Ilustración 97 - Estadísticas del tablero resuelto

- **Volver**: Vuelve al menú inicial.

Editor

En el momento en el que se hace click en el botón de aceptar y nos encontramos en el submenú editor, se abre el editor y se sitúan los vehículos de la tarjeta sobre el tablero (excepto si es un tablero nuevo) y se muestran los vehículos disponibles.

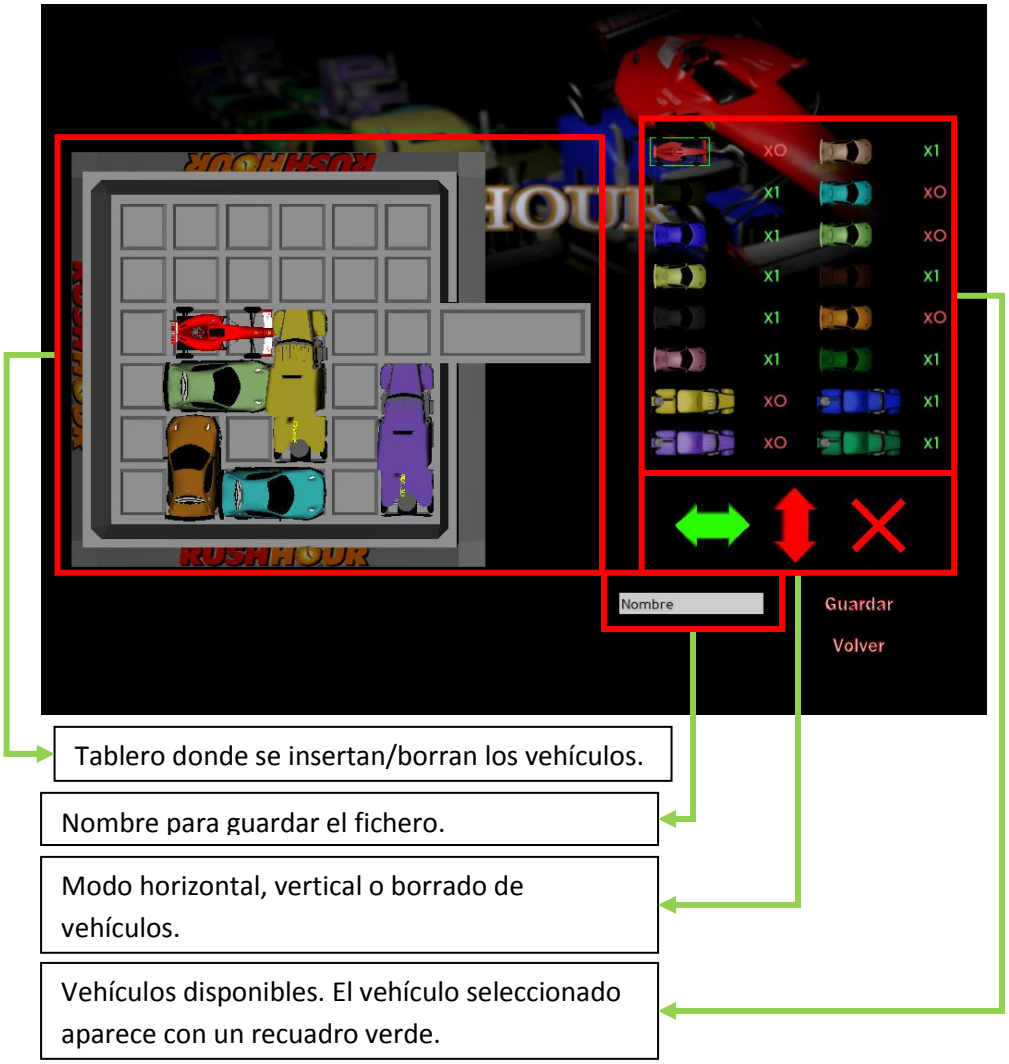


Ilustración 98 - Editor de tableros

- **Guardar**: Se comprueba si el tablero está correctamente construido (al menos está puesto el coche principal) y se guarda siempre y cuando el nombre no esté usado previamente.
- **Volver**: Vuelve al menú inicial.

En el editor también se puede girar el tablero manteniendo el botón izquierdo del ratón presionado y moviéndolo. Una vez soltado el botón volverá a su posición original.

El editor, además, indica si un vehículo puede o no ponerse en la posición y orientación indicada:

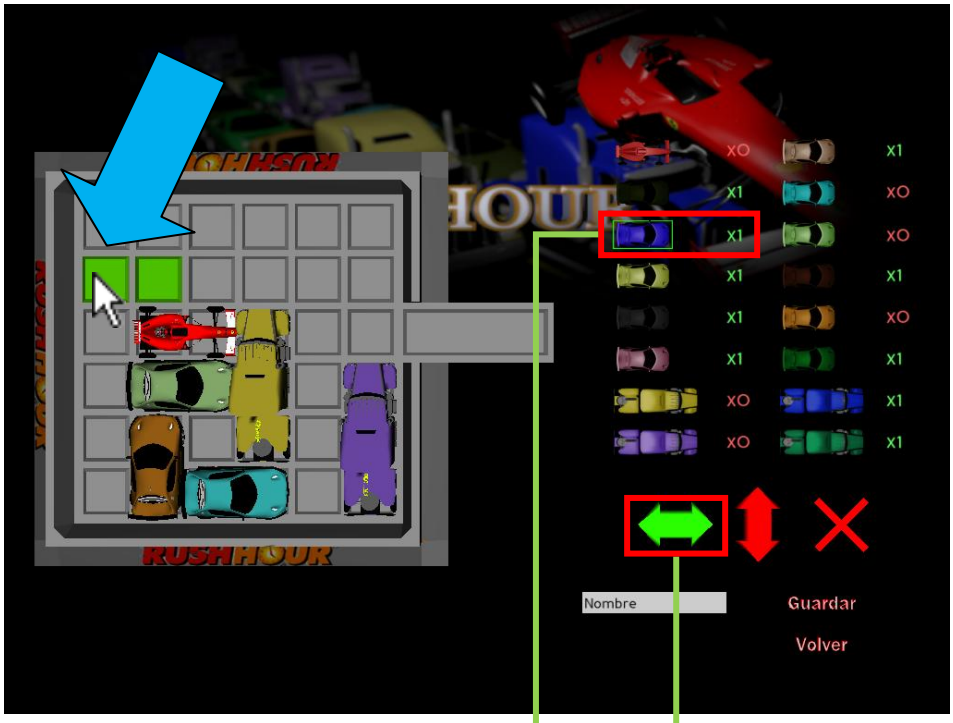


Ilustración 99 - El vehículo se puede insertar

Ilustración 100 - El vehículo no se puede insertar

Ver coches

Este apartado está hecho como un extra para poder ver los modelos de los vehículos con mayor detalle.

Ilustración 101 - Visor de vehículos

- **Flecha izquierda/derecha**: Cambia el vehículo.
- **Mover ruedas**: Activa/desactiva el movimiento de las ruedas.
- **Botón +/-**: Aumenta/disminuye la velocidad de rotación de la peana.
- **Volver**: Vuelve al menú inicial.

Anexo D

Referencias y bibliografía

1. http://en.wikipedia.org/wiki/Fifteen_puzzle
2. http://en.wikipedia.org/wiki/Rush_Hour_(board_game)
3. http://en.wikipedia.org/wiki/Reversi
4. http://en.wikipedia.org/wiki/Chess
5. http://en.wikipedia.org/wiki/Forza_Motorsport_4
6. http://en.wikipedia.org/wiki/Ace_Combat_6:_Fires_of_Liberation
7. http://en.wikipedia.org/wiki/Mars_Pathfinder
8. http://en.wikipedia.org/wiki/Nob_Yoshigahara
9. http://en.wikipedia.org/wiki/Thinkfun
10. Stuart Russell & Peter Norvig, Artificial Intelligence, A Modern Approach, editorial Prentice Hall
11. Stuart Russell & Peter Norvig, Artificial Intelligence, A Modern Approach, editorial Prentice Hall
12. http://www.blender.org
13. http://www.gimp.org
14. http://www.panda3d.org
15. http://www.gedit.org
16. http://www.swig.org

Otra bibliografía:

- Manual Panda3D
- API de Panda3D para Python
- Manual SWIG para Python
- Web del grupo de inteligencia artificial de la UC3M
- Mark Summerfield, Python 3, editorial ANAYA
- Miguel Angel Acera García, C/C++, editorial ANAYA